ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE COMPUTADORES

IMPLEMENTACIÓN Y ANÁLISIS DE METAHEURÍSTICOS DE OPTIMIZACIÓN COMBINATORIA SOBRE REDES P2P

IMPLEMENTATION AND ANALYSIS OF COMBINATORIAL OPTIMIZATION METAHEURISTICS ON P2P NETWORKS

Realizado por:
FRANCISCO HEREDIA MORENO

Tutorizado por:
Dr. CARLOS COTTA PORRAS

Departamento:
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, FEBRERO 2017

Fecha defensa:
El Secretario del Tribunal

Resumen

La mayoría de los problemas de optimización combinatoria tienen un difícil tratamiento computacional porque los algoritmos que los solucionan lo realizan en tiempo no polinómico, puesto que existe una relación de estos con la clase de problemas NP-ardua. Debido a que esta situación parece insalvable, a no ser que P=NP, se han desarrollado una serie de métodos genéricos que se aproximan a las soluciones del problema en un tiempo asequible; llamados metaheurísticos. El representante de los problemas de optimización combinatoria es el problema del viajante de comercio (TSP, siglas en inglés).

Los problemas complejos suelen tratarse a través de la computación distribuida. Estos sistemas suelen tener un coste muy elevado por la gran envergadura de su infraestructura. Sin embargo, las redes P2P han posibilitado desplegarlos fácilmente a bajo coste, dando origen a la computación voluntaria.

Veremos la implementación de algunos metaheurísticos para el problema del viajante de comercio para ser ejecutados en un simulador de redes P2P para observar su comportamiento.

Los experimentos, que se han ejecutado en el simulador, han generado una serie de datos, los cuales se han incorporado a este trabajo para su análisis.

Palabras clave: optimización combinatoria, problema el viajante de comercio, redes P2P, simulador, PeerSim.

Abstract

The travelling salesman problem (TSP) is one of the most extensively studied problems in optimization. As it is also the case of many other interesting combinatorial optimization problems, it falls within the NP-hard class, and solving this kind of problems to optimality poses a formidable challenge. No polynomial-time algorithm is known for this purpose and therefore unless P=NP, metaheuristics are often the weapon-of-choice to tackle them. Suchan approach is not guaranteed to find an optimal solution, but it is expected to run quickly (in polynomial time). Still, more and more computational resources may be required as the size of the problem scales up.

Distributed computing is an excellent tool for tackling complex problems. While traditional distributed systems require a large dedicated infrastructureand can therefore be expensive, new systems are emerging nowadays.
For example,P2P networks provide a scalable, low cost alternative. In this work, some metaheuristics have been developed for the TSP to be run on a P2P networksimulator (PeerSim) in order to study their behavior. The experiments, whichhave been run in this simulator, have generated an extensive data set, which have been incorporated into this work for analysis.

Keywords: Combinatorial optimization, travelling salesman problem, P2P networks, simulator, PeerSim

Índice de contenido

CAPÍTULO 1. INTRODUCCIÓN.

1.1.- MOTIVACIÓN.

La computación distribuida es un sistema informático complejo donde varias computadoras son interconectadas entre sí, con el fin de aumentar la capacidad de cómputo para la resolución de grandes problemas.

Existen varios modelos de computación distribuida, los cuales tienen en común su alto costo y complejidad. Pero con el nacimiento de Internet y las redes entre iguales (P2P), pronto surgió un modelo de este paradigma de computación donde ordenadores independientes entre sí y con características heterogéneas entre ellos, podrían unirse y prestar tiempo de computación para la resolución conjunta de un problema grande.

Esto ha dado al nacimiento de la *computación voluntaria*, donde cualquier persona con su ordenador puede a voluntad prestar capacidad de cómputo durante un tiempo deseado para la resolución de un problema determinado. Así de esta manera, numerosas computadoras distintas entre sí, pueden unirse a nivel global para una misma tarea.

Un número importante de proyectos, sobre todo científicos, han optado por la computación voluntaria para disponer de un centro de cálculo.

Gran parte de los problemas de optimización combinatoria suelen pertenecer a la clase NP-ardua y ser resueltos por algoritmos con una complejidad temporal no polinómica, ya que, explorar todo el espacio de soluciones requerirá de un elevado tiempo de computación. Por lo que, a partir de esta premisa, se han desarrollado una serie de métodos que intentan evitar o saltar una parte del espacio de soluciones para acotar el tiempo de cómputo, aunque asuman una pérdida de calidad en la solución.

Algunos de estos métodos son generales y de común aplicación a la mayoría de problemas de optimización combinatoria, son llamados algoritmos *metaheurísticos*.

El problema del viajante de comercio (*Traveling Salesman Problem – TSP*) es uno de los más conspicuos y estudiados miembros del rico conjunto de problemas de optimización combinatoria. El TSP se puede enunciar de la siguiente forma: existe un viajante de comercio (representante, vendedor) que debe realizar una visita a *n* ciudades, que están conectadas entre sí, pasando por todas ellas una sola vez, sin repetir ninguna, y volver a la ciudad de partida realizando el camino más corto para ello.

Pese a que, a priori, pueda parecer tener una solución sencilla y, en teoría, se conoce como resolverlo; en la práctica, presenta un gran problema porque el tiempo

de computación para aplicar las soluciones conocidas para el problema es muy elevado, incluso para un número pequeño de ciudades. Se ha podido demostrar que la resolución óptima de este problema es NP-ardua, por tanto, las técnicas de resolución exacta conocidas tendrán en el peor caso un comportamiento no-polinómico con relación al tamaño de la instancia del problema.

La gran dificultad de resolución de este problema ha hecho que los esfuerzos se hayan orientado más hacia la vertiente heurística.

El estudio de este problema ha atraído a numerosos investigadores de muy diversos campos, tales como las Matemáticas, la Investigación Operativa, la Física, etc pues el TSP tiene numerosas aplicaciones prácticas en múltiples áreas donde haga falta minimizar un recorrido o similar. Es por ello, que el TSP es un problema que ha tratado de resolverse con gran ahínco.

1.2.- OBJETIVOS DEL PROYECTO.

Este Trabajo Fin de Grado tiene como objetivo el diseño y la implementación de varias metaheurísticas, como la búsqueda dispersa, junto a las estructuras de datos necesarias para la manipulación de los problemas y los sistemas planteados, adaptándolas a ser ejecutadas en entornos distribuidos como las redes P2P.

Para el desarrollo de las implementaciones se creará una red P2P de forma simulada mediante los distintos simuladores existentes, que den base de cómputo distribuido a las implementaciones realizadas.

Una vez alcanzados estos objetivos, y funcione el sistema planteado. Se realizarán distintos análisis de rendimientos de las metaheurísticas contempladas.

1.3.- ESTRUCTURA DE LA MEMORIA.

La memoria de este proyecto se divide en 8 capítulos y una bibliografía.

- Capítulo 1. Introducción. Es el capítulo actual y en el se describe brevemente y se hace una introducción del presente trabajo fin de grado. Además se resume el contenido de la presente memoria.
- Capítulo 2. El problema del viajante de comercio. En este capítulo se define formalmente el problema del viajante de comercio; para ello, al principio del capítulo, se presenta ligeramente los conceptos matemáticos

que intervienen en el problema. Se listan los tipos del problema -que se resuelven con esta biblioteca- y las técnicas para solucionar el problema. Además, se expone la complejidad computacional del TSP, una breve introducción histórica del TSP y sus aplicaciones prácticas.

- Capítulo 3. Redes P2P. Al ser conceptos muy extensos y complejos, se hace una introducción a la computación distribuida, las redes de entre pares y la computación voluntaria. El protocolo Newscast es usado en redes P2P para implementar una arquitectura no estructurada y la comunicación entre nodos. Además se explica el funcionamiento del simulador de redes P2P, llamado, PeerSim.

- Capítulo 4. Implementación de los metaheurísticos. Este capítulo se describe el conjunto de paquetes y clases que se han implementado para la realización de este trabajo fin de grado y conseguir la ejecución de los metaheurísticos objeto de este trabajo en el simulador PeerSim.

- Capítulo 5. Evaluación de las simulaciones. Se exponen los resultados que se han obtenido en el análisis efectuado a las heurísticas que se han empleado.

- Capítulo 6. Conclusiones finales. En este capítulo se detalla en conjunto los resultados de la realización de esta trabajo y las posibles mejoras de esta en el futuro.

- Capítulo 7. Bibliografía. Se describe el material documental utilizado.

- Capítulo 8. Anexo I. Instalación y ejecución. Se aporta una pequeña guía o manual para la instalación del entorno de desarrollo que se ha usado en la elaboración de este trabajo, para que pueda reproducirse sin problemas por otra persona, y las órdenes de para la ejecución del simulador con las soluciones desarrolladas.

CAPÍTULO 2. EL PROBLEMA DEL VIAJANTE DE COMERCIO.

2.1.- ENUNCIADO DEL PROBLEMA DEL VIAJANTE DE COMERCIO

El problema del viajante de comercio consiste en que: un viajero tiene que realizar un ruta por un número determinado de ciudades, para ello, debe visitar cada ciudad exactamente una vez y debe regresar al punto de partida. Se le da un costo para viajar entre todos los pares de ciudades. El viajero debe planear su ruta para que, además de visitar cada ciudad una vez y regresar a la ciudad de partida, el costo total de la ruta sea el de menor coste o mínimo.

Más formalmente se define el problema como sigue:

Instancia:

1. Sea C un conjunto de *m* ciudades.
2. Sea $d(c_i,c_j) \in \mathbb{R}^+$ la distancia entre dos ciudades, $c_i, c_j \in C$.
3. Sea $B \in \mathbb{R}^+$.

Pregunta:

¿Hay una ruta de C que tenga longitud igual a B o menor, por ejemplo, una permutación $< c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(k)}, c_{\pi(k+1)}, \ldots, c_{\pi(m)} >$ de C, tal que:

$$(\sum_{i=1}^{m} d(c_{\pi(i)}, c_{\pi(i+1)})) + d(c_{\pi(m)}, c_{\pi(1)}) \leqslant B \ ?$$

Este mismo problema puede formularse, también, desde distintos enfoques:

a) Como un problema de optimización combinatoria:

Instancia: Sea G un grafo ponderado, siendo los pesos números reales positivos.

Solución factible: Un ciclo que pasa a través de cada vértice de G exactamente una vez; por ejemplo, un ciclo hamiltoniano.

Función Objetivo: Para un ciclo hamiltoniano H de G se define $d(H) = \sum d(e)$ donde $e \in H$

Solución Óptima: Encontrar un ciclo hamiltoniano de costo mínimo.

b) Como un problema de decisión:

Intancia: Un grafo G ponderado y un hipotético costo D.

Pregunta: ¿Tiene G un ciclo hamiltoniano?

c) Como un problema euclidiano:

Instancia: Sea G un grafo ponderado, siendo los pesos números reales positivos y donde la desigualdad del triángulo se satisface: para todos los vértices u, v y w se cumple que $d(u,v) \leqslant d(u,w) + d(w,v)$.

Solución factible: Un ciclo hamiltoniano.

Función Objetivo: Para un ciclo hamiltoniano H de G se define $d(H) = \sum d(e)$ donde $e \in H$.

Solución Óptima: Encontrar un ciclo hamiltoniano de costo mínimo.

El problema euclidiano es un caso particular de a)

2.2.- SOLUCIONES PARA EL PROBLEMA DEL VIAJANTE DE COMERCIO.

Si bien las variantes del TSP las hemos visto desde la definición como grafo del problema del viajante de comercio por su facilidad visual y de exposición; este nuevo aspecto, el de las soluciones, serán vistas de un modo más eficaz a través de la definición del TSP como un problema de optimización combinatoria y/o un problema de decisión. Recordemos que un problema de optimización combinatoria es aquel que consiste en asignar valores a un conjunto de variables de manera que una función objetivo sea optimizada y, además, un conjunto de restricciones es satisfecho. En el caso concreto del TSP es encontrar una permutación del conjunto de ciudades, teniendo como función objetivo a encontrar un valor de la longitud del camino sea mínimo y donde hay que satisfacer las siguientes restricciones: recorrer todas las ciudades sin repetir ninguna y volver a la misma ciudad desde la que se partió.

El problema del viajante de comercio tiene un enunciado sencillo, de fácil comprensión y, a priori, puede parecer que la solución al problema es igualmente elemental; sin embargo, muchos investigadores durante mucho tiempo han

tratado de solucionarlo de diversas maneras generando distintos métodos o aplicando técnicas existentes para ello; condicionados siempre por la complejidad computacional del problema. Tanto es así, que los métodos que se desarrollan para problemas de optimización combinatoria son primeramente ensayados en el problema del viajante de comercio para comparar así sus resultados con otros ya existentes; es decir, el TSP es un verdadero banco de pruebas y cualquier procedimiento es probado en el TSP para ser validado.

Los métodos para optimización combinatoria, y específicamente para el TSP, en su conjunto pueden clasificarse en *métodos completos*, los cuales encuentran solución óptima a una instancia del problema pues hacen una búsqueda global en el espacio de soluciones, y en *técnicas heurísticas* que dan a lugar a una "buena" solución pero no es la óptima, entendiendo como "buena" a una solución cuya calidad o que el tiempo de generación de la misma está acotado, siendo ésta una búsqueda local de las soluciones.

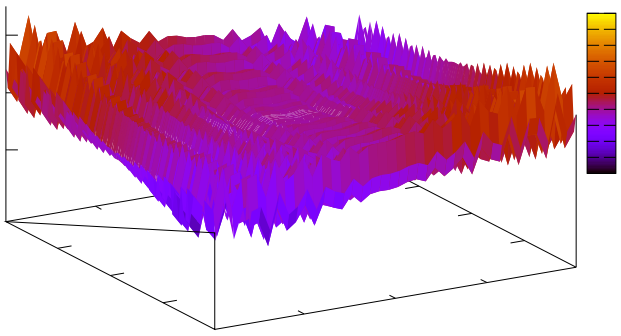


Fig. 2.12. Espacio de soluciones de una función dada.

La representación gráfica del espacio de soluciones de una función objetivo cualquiera es representado por la figura 2.12; en dicha gráfica observamos que sobre el eje de ordenadas existen numerosos picos, tantos positivos como negativos, los cuales corresponden a máximos y mínimos respectivamente. Todos estos óptimos son locales respecto a su entorno, y el de mayor valor positivo de entre todos ellos será el máximo global y, a la inversa, el mayor valor negativo corresponderá con el mínimo global.

Entonces vemos que un método completo ha de tener como estrategia el conocer todos los puntos de interés y compararlos para averiguar cuál es el

óptimo buscado, de ahí se dice que realiza una búsqueda global. Por ejemplo, si la gráfica de la figura 2.12 se trasladará a una cartulina en relieve, una búsqueda global sería pasar la mano por todo el área de la cartulina y quedarnos con el pico más alto.

Otros métodos, basados en la búsqueda local, en su exploración pueden quedar “encajados” en el área o entorno de un óptimo local, siendo incapaces de recorrer más porción del espacio de soluciones y, por tanto, no encontrar al óptimo global. Sólo el azar o algunas técnicas específicas para el problema serán capaces de centrar la búsqueda del algoritmo en un entorno cercano al óptimo global, llegándolo a encontrar -en escasos casos- o acercarse al punto de interés que será el objetivo de los algoritmos de búsqueda local. Siguiendo con el ejemplo mencionado en la búsqueda global, si tuviéramos es cartulina en relieve que representa a la función, un método de búsqueda local sería soltar una canica en un punto al azar del área de la cartulina. La canica botará hasta quedarse encajada en alguno de los picos negativos, por tanto, habrá encontrado un mínimo local, y sólo la suerte hará que la canica haya quedado encajada en un mínimo global. Veremos que existen diversos métodos para hacer más determinista o, también, “inteligente” tanto el lanzamiento como el bote de la canica.

2.2.1.- MÉTODOS COMPLETOS.

Se llaman así a los métodos que realizan una exploración completa del espacio de soluciones, esto es, realiza una búsqueda global para encontrar un óptimo global al problema. Por su naturaleza son computacionalmente caros.

Técnicas exactas.

El TSP pertenece a la clase de problemas que tienen solución, por tanto, existen formas de averiguar la solución óptima de cualquier instancia del mismo. Al ser un problema de optimización combinatoria siempre seremos capaces de explorar el dominio de soluciones o generar todas las permutaciones posibles para una instancia dada y encontrar el óptimo. He aquí, pues, el primer método exacto que se puede aplicar al problema del viajante de comercio (generar todas las permutaciones posibles para una instancia dada) que suele denominarse de *fuerza bruta*, sin embargo, inherente al mismo problema de crear todas las permutaciones está el elevado coste computacional de este método, pues, para un problema de n ciudades, el número de permutaciones posibles es n!, haciéndose intratable ya para valores no muy grandes de n.

Lamentablemente, esta situación será una constante para cualquier técnica que aspire a solucionar de manera óptima el TSP, ya que, pertenece a la clase de problemas que son NP-ardua que, como veremos en el próximo apartado, tienen un tiempo de ejecución no polinómico. Además, del problema del elevado tiempo de ejecución, éstos métodos exactos suelen encontrar problemas con la memoria necesaria para ejecutar el algoritmo en cuestión, ya que, necesitan guardar todas las soluciones obtenidas para comparar cual de ellas es la mínima.

De todas formas, existen varias técnicas de la algoritmia que evitan explorar una parte del domino de soluciones que no van a dar con la solución buscada, y por tanto, creando una cierta mejora en el tiempo de computación; siendo éste, todavía, invariablemente no polinómico. A continuación veremos algunas de ellas:

- Programación dinámica.

  La programación dinámica es un técnica de la algoritmia que trata de optimizar el tiempo de ejecución de los problemas cuyas soluciones pueden ser expresadas recursivamente en términos matemáticos, y posiblemente sea la manera más natural de resolverlo mediante un algoritmo recursivo. Ya que, el método recursivo normalmente suele tener un orden de ejecución exponencial.

  La eficiencia de la programación dinámica consiste en resolver los subproblemas una sola vez, guardando sus soluciones en una tabla; así no hay que repetir cálculos si son necesarios en un futuro.

  Los problemas de optimización suelen ser resueltos eficazmente mediante esta técnica, ya que, este tipo de problemas suelen tener varias soluciones y hay que elegir cual es óptima, es decir, máxima o mínima.

  La programación dinámica se basa en el llamado principio de óptimo enunciado por Bellman en 1957, el cual dice: *“En una secuencia de decisiones óptimas toda subsecuencia ha de ser también óptima”.* Aunque parece una afirmación evidente es necesario comprobar que se cumple en cada caso, pues curiosamente no se cumple en el problema análogo al TSP, que trata de encontrar el camino de coste máximo entre dos vértices de un grafo ponderado.

  La aplicación de la programación dinámica en el caso del problema del viajante de comercio podría ser la siguiente: Si un recorrido es óptimo, también lo es un recorrido a una ciudad intermedia que haya en el

recorrido primero, es decir, un subrecorrido del primero. La ecuación de recurrencia que daría lugar al tratar de resolver el TSP mediante programación dinámica es esta: llamaremos $D(v_i,S)$ a la longitud del camino mínimo que partiendo del vértice v_i pasa por todos los vértices del conjunto S y vuelve al vértice v_i, la solución al problema del viajante de comercio vendrá dada entonces por $D(v_1,V-\{v_1\})$:

$$D(v_1,V-\{v_1\}) = \min_{2\leq k\leq n} \{L(v_1,v_k) + D(v_k,V-\{v_1,v_k\})$$

La ecuación de recurrencia dada es válida si partimos del primer vértice, v_1, es necesario generalizar esta ecuación de recurrencia para partir desde cualquier vértice, siendo trivial la misma, pero no es objetivo de este proyecto profundizar más en esta técnica.

Como conclusión, podemos ver que al final necesitaremos una tabla con n filas y 2^n columnas (el cardinal del conjunto V) para guardar las soluciones intermedias, por lo que, se hacen demasiadas consiguiendo, por tanto, una escasa mejora en la eficiencia respecto a un algoritmo para generar todas las posibilidades pasando de una complejidad factorial a una complejidad exponencial obtenida por esta técnica.

- Vuelta atrás.

  Esta técnica también es conocida por su nombre en inglés, *Backtracking*, y es ampliamente utilizada porque siempre puede ser aplicada a una gran cantidad de problemas -a todos los que sus soluciones puedan resolverse en etapas- y, por tanto, cuando otras técnicas fracasan, Vuelta atrás puede ser usada.

  La estrategia del diseño Vuelta atrás es la de explorar exhaustivamente todo el espacio conocido de posibles soluciones; de este modo, encontramos todas las soluciones y, en consecuencia, también la óptima.

  En su forma básica la Vuelta atrás se asemeja a un recorrido en profundidad dentro de un árbol, al que suele denominarse *árbol de expansión*; este árbol es sólo conceptual y no forma parte de la implementación, sin embargo, suele hacerse referencia a su organización para diferenciar las etapas de generación de la solución: niveles del árbol o las soluciones parciales o totales obtenidas: nodos del árbol. Cuando una rama del árbol no se puede completar, se llama nodo fracaso, se vuelve atrás -de ahí el nombre de la técnica- y se continúa ramificando por el

nodo padre factible más cercano para seguir buscando más soluciones.

En general, y en particular para el TSP se puede aplicar este diseño porque la solución se puede expresar como una n-tupla [x_1, x_2, …, x_n] siendo n el número de ciudades, y cada nodo tendrá un n-tupla parcial cuyo cardinal será el número de nivel en el que esté dicho nodo. Para el TSP un nodo fracaso sería aquel que produjera un ciclo sin haber recorrido todos las ciudades o que no permita volver a la ciudad de partida. Por tanto, una n-tupla siempre representará una solución parcial o total válida al problema. Por tanto, hay que definir unas restricciones simples que permitan detectar o evitar nodos fracasos; estas restricciones se dividen en explícitas e implícitas; en esto consiste la poca eficiencia que puede aplicarse a este diseño.

- Ramificación y poda.

  Más conocido en inglés, *Branch & Bound*, es una variante del diseño Vuelta atrás y la importancia para el TSP es que esta técnica se emplea para problemas de optimización y, normalmente, cuando interesa encontrar hasta el óptimo una instancia del TSP se recurre a algoritmos diseñados mediante esta técnica, en casos reales entre otros.

  Al igual que Vueltra atrás, esta técnica trata de explorar el espacio de soluciones mediante un árbol de expansión. Sin embargo, esta búsqueda, en vez de ser sistemática como en el caso de Vueltra atrás, viene definida por una serie de restricciones que evitarán explorar ramas del árbol que no conducen a una solución aceptable.

  En Vuelta atrás el recorrido del árbol de expansión se hacía en anchura para generar todas las soluciones posibles. En Ramificación y poda hay tres estrategias para recorrer o **ramificar** el árbol, estas son: en anchura (estrategia LIFO), en profundidad (estrategia FIFO) o utilizando el cálculo de funciones de coste para seleccionar el nodo más prometedor (estrategia del mínimo coste o LC).

  Además, de estas estrategia para ramificar el árbol de expansión, se establecen cotas para **podar** aquellas ramas que no conducen a la solución óptima del problema. Esto da a lugar a la definición de *nodo vivo* del árbol a aquel nodo que tiene posibilidades de ser ramificado y no ha sido podado aún. Así pues, todo algoritmo diseñado mediante esta técnica tendrá tres etapas: selección (un nodo de entre los nodos vivos), ramificar

el nodo seleccionado y poda de los nodos creados y que no conducirán a la solución buscada. En el desarrollo de estas etapas se encuentra la eficiencia de esta técnica que restringe el espacio de soluciones a explorar.

El diseño de un algoritmo mediante esta técnica para el problema del viajante de comercio admite diferentes estrategias de ramificación y poda, que explicarlas aquí se escapan del alcance de la presenta memoria.

Hasta aquí hemos descrito algunos modos de averiguar la solución óptima para TSP mediante técnicas exactas, existen otras, sobretodo basadas en la investigación operativa, como la programación lineal, método simplex o *Branch & Cut*, una variante de Vuelta Atrás también muy empleada en casos reales.

Algoritmos de aproximación.

Los algoritmos de aproximación usan técnicas sistemáticas cuyos resultados son aproximaciones del valor verdadero y se usan como alternativa a los métodos exactos o analíticos o, porque definitivamente, son la única forma de resolver el problema en cuestión. La mayoría de los algoritmos de aproximación provienen de la investigación operativa como la teoría de colas, algoritmos probabilísticos o métodos numéricos.

Un ejemplo sencillo de algoritmo aproximado para el TSP es:

1. Generar un árbol de expansión mínimo para G, siendo G el grafo de una instancia para el TSP.
2. Devolver el circuito resultante de recorrer el árbol en preorden.

Sin embargo, está demostrado que para el problema del viajante de comercio no existe un método aproximado eficiente, a menos, que P=NP lo cual la comunidad científica no cree que sea cierto. Excepcionalmente pueden encontrarse algunos si la función de coste cumple la desigualdad del triángulo, siendo estos del tipo 2-aproximaciones o aproximaciones de radio 2 como mínimo. Es decir, no se encontrarán aproximaciones de radio 1.

2.2.2.- TÉCNICAS HEURÍSTICAS.

En general, y para problemas como el viajante de comercio, sería deseable que existiera un algoritmo que para un tiempo de computación razonable

obtuviera el resultado óptimo para un instancia determinada del TSP. Como hemos mencionado anteriormente, está demostrado que esto no puede ser porque el problema del viajante de comercio es un problema NP-arduo.

Un algoritmo heurístico es aquel que no puede asegurar nada sobre el resultado que obtiene, aunque generalmente bueno o aceptable, pero su tiempo de ejecución es muy inferior respecto a los métodos completos aunque, tampoco, pueden asegurar nada sobre el tiempo de computación. Esta incertidumbre, en cuanto a la calidad de los resultados y/o al tiempo de ejecución, no suele presentarse para todas las instancias del problema objetivo; sino que para un espacio reducido y acotado de los datos de entrada se producen resultado muy malos o los tiempos de ejecución se disparan; sin embargo, para la gran mayoría de instancias el heurístico tiene un comportamiento aceptable.

La palabra heurística etimológicamente proviene de la palabra griega *heuriskein* que significa encontrar o descubrir. Se suele definir a los algoritmos heurísticos como aquellos que se asemejan a la forma de solucionar los problemas como lo realizan los humanos, esto es, en cuanto a la creatividad para desarrollar soluciones fuera de un ámbito científico o teórico, en contraposición a los métodos completos que se basan en el análisis matemático o la síntesis de soluciones. Suelen basarse en experiencias, métodos u observaciones de fenómenos naturales, los cuales pueden ser aplicados específicamente para el problema que tratan de resolver o ser una estrategia general aplicable a muchos o todo tipo de problemas. Un ejemplo muy extendido de un heurístico básico utilizado ampliamente en muy diversas áreas y en la vida cotidiana en general como forma de aprendizaje, es el método de *prueba y error*.

El crecimiento en el desarrollo de procedimientos heurísticos es tan espectacular en los últimos años que incluso en 1995 se creó una publicación dedicada íntegramente a este tipo de investigaciones, llamada *Journal of Heuristics.*

Normalmente el uso de heurísticos para resolver un problema se realiza dadas las siguientes condiciones:

- Existe un método completo que resuelve el problema pero computacionalmente es muy costoso.
- No se conoce un método completo para el problema.
- El heurístico sea más flexible para incorporar ciertas restricciones que sean difíciles de modelar que los métodos completos.

- El método heurístico forma parte de una estrategia global para resolver un problema donde es utilizado o para generar una buena solución que se usará como punto de partida para un método exacto o participa en un paso intermedio de un método completo.

Principalmente estos métodos suelen ser aplicados en problemas de optimización; pertenecen al orden de computación polinómico y, normalmente, se dividen en deterministas o no deterministas.

La mayoría de esta clase de algoritmos suelen establecer una cota en cuanto al tiempo que proporcionan una solución e, igualmente, en cuanto a la bondad de la misma. Para ello, existe una métrica teórica estándar para medir la calidad de un algoritmo heurístico, que es el cociente para el peor de los casos que indica que tan cerca del valor óptimo están las soluciones encontradas por este tipo de algoritmos. Dicha cercanía puede ser expresada como el cociente del valor obtenido por el heurístico sobre el valor de la solución óptima; dicho cociente puede ser expresado de la siguiente manera:

- Sea I una instancia para un problema que resuelve un algoritmo heurístico.
- Sea OPT(I) el valor óptimo de la función objetivo para la instancia I.
- Sea V(I,A(I)) el valor obtenido por el heurístico A al evaluar la instancia I en la función objetivo.

$$C_A = \left\{ \frac{V(I, A(I))}{OPT(I)}, \frac{OPT(I)}{V(I, A(I))} \right\}$$

Nótese que C_A es siempre menor o igual que 1.

Por ejemplo, se sabe que cuando el costo de las aristas satisface la desigualdad del triángulo en el problema del viajante de comercio, existe un algoritmo simple que siempre produce un camino de longitud a los más dos veces la longitud del camino óptimo. Y esto tiene un tiempo de ejecución $O(n^2)$. Un algoritmo mejorado que tenga $O(n^4)$ siempre encuentra un camino de longitud 2/3 el valor de la longitud del camino óptimo. Y hasta la fecha no se conoce un algoritmo mejor para el problema.

Para el viajante de comercio se han desarrollado algunas heurísticas específicamente para resolver este problema en cuestión que suelen obtener muy buenos resultados, fruto de la fama de este problema entre la comunidad científica encargada de estudiar los problemas de optimización combinatoria.

Los cuales algunos de ellos son los que forman la biblioteca desarrollada para este proyecto.

Para el TSP los heurísticos específicos se clasifican en dos tipos:

a) Heurísticos de construcción.

   Son algoritmos que reciben como entrada una instancia determinada para el problema del viajante de comercio, mediante a algún criterio selecciona las ciudades que se van a añadir al camino que se construye. Al final la salida del algoritmo es un camino que cumple las condiciones del problema y cuya valor de la longitud se aproxima al óptimo. Estos algoritmos suelen diferenciarse entre sí por el tipo de criterio que se utiliza para seleccionar los nodos candidatos a formar el camino.

b) Heurísticos de mejora

   Estos algoritmos toman como entrada un camino posible para una instancia del TSP. El heurístico de mejora tiene un método para alterar o intercambiar el orden de las posiciones de las ciudades en el camino dado. De esta manera, crea un nuevo camino cuya longitud es menor a la longitud del camino dado como entrada, y por tanto, acercándolo aún más al valor del camino óptimo para dicha instancia del TSP. En aras de obtener los mejores resultados, el camino dado como entrada al heurístico de mejora suele ser generado por un heurístico de construcción, así se parte desde el inicio con una longitud del camino para la instancia del problema dado ya cercana al óptimo.

Otras heurísticas

Dada la forma de generación de métodos heurísticos, existe una gran cantidad de ellos y su forma de operación es igualmente diversa; por lo que, su clasificación es un tanto complicada. Además de heurísticos, también se hablan de metaheurísticas la cuales se pueden definir como aquellos procedimientos que proporcionan un método general y abstracto para resolver problemas.

Este campo es tan amplio que sólo vamos a mencionar a algunos que se han utilizado para resolver el TSP aunque no han sido diseñados específicamente para resolverlo pues son métodos generales que se aplican a otros muchos problemas:

- Búsqueda local.

  Generalmente los resultados dados por los heurísticos de construcción son

de una calidad moderada. Estos métodos parten de estas soluciones para mejorarlas, es decir, se tratan de encontrar un óptimo partiendo ya de un lugar concreto del espacio de soluciones. Por tanto, aspiran a encontrar un óptimo local; el cual puede ser el óptimo global en algunos casos.

Como vemos estos métodos de búsqueda local se asemejan a la definición de heurísticos de mejoras dada anteriormente y es que, efectivamente, es así pues los heurísticos de mejoras para el TSP son algoritmos de búsqueda local. Los algoritmos de búsqueda local suelen ser diseñados específicamente para el problema en cuestión que tratan de resolver.

- Procedimientos aleatorizados

  Con los heurísticos que hasta ahora hemos visto, vemos que la estrategia normal para el TSP es generar una solución con un algoritmo de construcción y, posteriormente, usar un algoritmo de mejora para disminuir el valor del camino. Sin embargo, en algunos casos se corre el riesgo que estos métodos son sean capaces de avanzar en el espacio de soluciones y queden encajonados en un área o entorno de dicho espacio; se habla entonces de miopía del método debido a que son deterministas.

  Con el fin de evitar esta limitación y permitir que los algoritmos salten a otros lugares del espacio de soluciones se aplican un factor indeterminista como puede ser el introducir en algunas de las fases valores aleatorios.

- GRASP

  Este nombre es la abreviatura de las palabras inglesas Greedy Randomized Adaptative Search Procedures *(Procedimientos de búsqueda voraz aleatoria adaptativa)*. Se basan en construir una solución a través de una estrategia voraz y luego mejorarla con una búsqueda local. El factor aleatorio es introducido por el algoritmo voraz, el cual en vez de elegir en cada iteración un sólo elemento, elige uno al azar de una lista de elementos. El término adaptativo se refiere al hecho de que los beneficios a cada elemento son actualizados en cada iteración de la fase de construcción para reflejar los cambios producidos por selecciones previas.

- Búsqueda Tabú

  Pertenecen a la clase de métodos de búsqueda local y tratan de aumentar el rendimiento de este tipo de algoritmos proporcionándoles una estructura de memoria para guardas potenciales soluciones que pueden

ser marcadas como tabú si el método así lo determina y no volver a buscar en ellas.

- Templado simulado

  Es una metaheurística muy utilizada, en inglés *simulated annealing (SA)*. Se inspiró en el proceso de templado del acero que consiste en calentar y luego enfriar controladamente el acero para aumenta el tamaño de los cristales y reducir sus defectos.

  Este algoritmo en cada iteración considera a algunos vecinos que se encuentran el estado s y decide probabilísticamente si pasan al estado s' o se quedan en el actual estado s. Las probabilidades se escogen para que el sistema tienda a un estado de energía menor. Esto se repita hasta que se encuentra una solución aceptable o a algún criterio de parada del algoritmo. Los vecinos de cada estado se eligen según el tipo de problema a resolver. Las variables a modelar en este método es la probabilidad de cambiar de estado y la velocidad de enfriado.

- Métodos evolutivos

  Estos métodos se inspiran en la naturaleza y en especial a la evolución biológica de las especies o seres vivos. También suelen llamarse métodos poblacionales porque a cada solución se le llama individuo y a un conjunto de ellas población. Parten de una serie de individuos los cuales representan posibles soluciones al problema, estos individuos interactúan (operador)-se mezclan, compiten entre sí, mutan, desaparecen- de tal manera que las más aptas son capaces de prevalecer en el tiempo, evolucionando hacia mejores soluciones cada iteración.

  Principalmente son usados cuando se necesita explorar un espacio de soluciones muy extenso y se clasifican como una rama de la inteligencia artificial.

  Dentro de los métodos evolutivos existen una gran cantidad de algoritmos que finalmente se han clasificado como evolutivos pero que originalmente fueron diseñados por separados, independientes uno de otros y por motivaciones distintas. Se pueden citar entre otros:

  - Algoritmos genéticos.
  - Evolución simulada.

- Algoritmos meméticos.
- Re-encadenamiento de trayectorias
- Búsqueda dispersa
- Colonias de hormigas
- Métodos evolutivos basados en modelos probabilísticos.

- **Redes neuronales artificiales**

  Es un modelo computacional inspirado en la simulación del funcionamiento de la neurona biológica y en la forma de conectarse entre ellas, partiendo de la evidencia anatómica y fisiológica de la misma. Es otra rama de la inteligencia artificial.

  Básicamente la idea es que si el cerebro es un órgano de los seres vivos que sirve para resolver problemas, entonces se puede crear un modelo computacional que simula su funcionamiento y pueda ser usado para este mismo propósito. Evidentemente, esta idea es un tanto utópica en cuanto a simular un cerebro completo con todas sus funcionalidades biológicas pero sí se consigue una simulación o modelizar una parte o conjunto de neuronas conectadas entre sí que resuelven problemas.

  Existen diversos modelos para crear redes neuronales fruto de muchos investigadores de diversas áreas; algunas de ellas son: el perceptrón, redes de Hopfield, retropropagación, memorias asociativas, mapas autoorganizados, etc

  En general, las redes neuronales no son la mejor forma de resolver un problema, pues normalmente existe un algoritmo mucho más eficiente, pero pueden ser aplicadas a una gran cantidad de problemas donde en algunos casos producen resultados muy ventajosos y ser implementadas de muy diversas maneras.

  Para el problema del viajante de comercio también se han usado redes neuronales con más o menos éxito. Especialmente Redes de Hopfield con unos resultados normales pero de interés por su ingeniosa implementación y Redes de Kohonen con unos resultados muy buenos, esta vez.

2.3.- COMPLEJIDAD COMPUTACIONAL DEL TSP.

Dentro de la teoría de la computación existe una rama, llamada complejidad computacional, que se encarga del estudio del uso de los recursos del computado por parte de un algoritmo, esto es, comportamiento y rendimiento de los algoritmos; principalmente, en dos aspectos: el tiempo que tarda el algoritmo en ejecutarse, o complejidad temporal, y el espacio de memoria necesario para su ejecución, o complejidad espacial.

Las clases de complejidad son conjuntos de problemas que comparten un mismo orden de complejidad. Hablaremos aquí sobre la clase P, NP y NP-ardua.

La clase de complejidad P son los que se ejecutan en un tiempo polinómico en una máquina de Turing, es decir, el tiempo de ejecución de un algoritmo que pertenece a P no se ve gravemente aumentado si los datos de entrada aumentan de tamaño. En un sentido práctico se dice que los problemas P son tratables porque siempre pueden ser abordables realmente por un computador en una situación real.

La clase de complejidad NP son los problemas cuyo tiempo de ejecución es superior a la polinómica. La N significa No-determinista y la P es polinómico. Por tanto, cuanto más aumente de tamaño los datos de entrada, los tiempos de ejecución se disparan extremadamente (de forma exponencial, factorial, …); por lo que hace que los problemas que pertenecen a NP se les consideren intratables a efectos prácticos.

Los problemas que pertenecen a la clase NP-ardua o NP-completos son aquellos problemas NP que son aún más difíciles de resolver en cuanto a su complejidad computacional. Esto es, son problemas muy intratables porque a pequeñas variaciones del tamaño de los datos de entrada, los tiempos de ejecución aumentan tremendamente. Por lo que si existe un algoritmo que los resuelva, éste es tan exageradamente ineficiente que no permite se usado en la práctica.

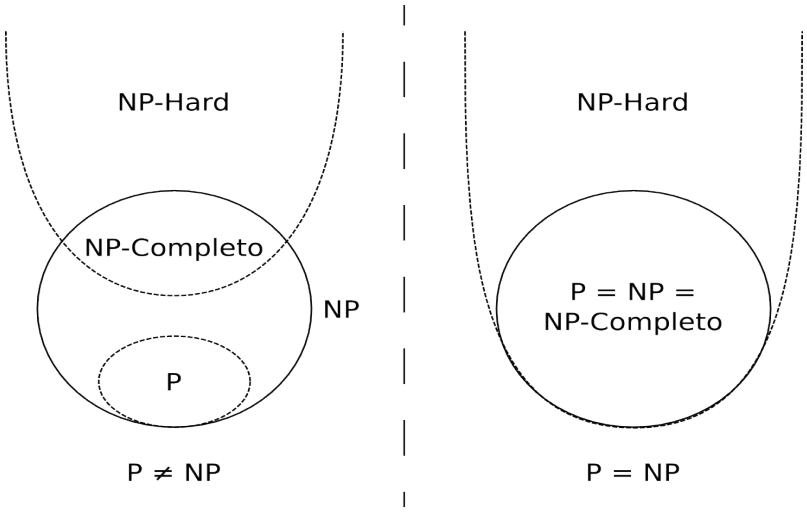


Fig. 2.13 Diagrama de Venn con la clasificación de la familia de problemas.

Vemos que en la figura 2.13 que la clasificación es de una manera u otra dependiendo si los conjuntos P=NP o no son iguales. En realidad, esta es una de las preguntas más importante a la que se ve sometida la ciencia de la computación, tanto es así que el Clay Mathematics Institute ofrece una recompensa a quien sea capaz de responderla. Pese a que no existe la certeza, la comunidad científica está ampliamente convencida de que $P \neq NP$.

Como hemos mencionado en apartados anteriores el problema del viajante de comercio pertenece a la clase de complejidad NP-ardua o NP-completo, y pese a que existen algoritmos que lo resuelven hasta el óptimo, éstos son intratables.

La demostración de que el TSP pertenece a la clase de problemas NP es derivada de otras demostraciones de problemas también NP, veremos a continuación un diagrama que mostrará gráficamente los problemas que intervienen en la demostración para el TSP.

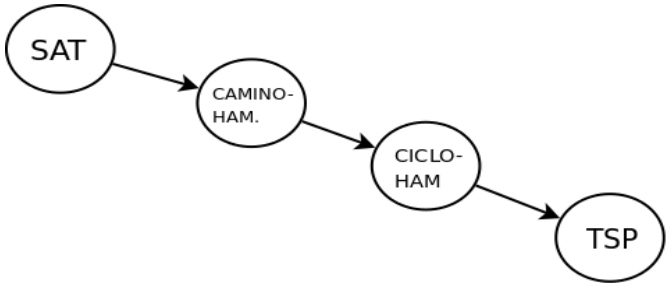


Fig. 2.14. Diagrama de problemas NP.

Describiremos que problemas son anteriores al TSP y asumiremos que está demostado que son NP y, por tanto, sólo mostraremos la demostración final para

el problema del viajante de comercio.

- SAT: Este en un problema de la lógica proposicional y se enuncia de la siguiente manera, teniendo una fórmula proposicional que es una fórmula bien formada (f.b.f) determinar si es satisfacible. Este problema está demostrado que es NP-arduo. Tiene que ver con el TSP porque las soluciones para el problema del viajante de comercio pueden expresarse como una t-upla, siendo ésta una fórmula proposicional bien formada.
- Camino Hamiltoniano, este problema es encontrar un camino que recorra todos y cada uno de los vértices de un grafo. Este camino puede ser expresado como una fórmula proposicional bien formada, de ahí se llega a la conclusión que este problema también es NP-arduo pues es una derivación del problema SAT.
- Ciclo Hamiltoniano, este problema es encontrar un camino hamiltoniano y volver al primer nodo del camino. Si encontrar un camino ya es un problema NP-arduo, sólo nos queda demostrar que el camino de regreso al punto de partida también lo es. Efectivamente, se cumple que encontrar un ciclo hamiltoniano también es NP-completo.

Entonces enunciamos el siguiente teorema: **El problema TSP es NP-arduo.**

- Demostración

Veamos que el problema $Ciclo\,Hamiltoniano \leq^{p} TSP$

Sea G=(V,E) un grafo no dirigido, con V={1, ..., n}. Sea C=V y d : C x C → N definida así:

$$d(i,j) = \begin{cases} 0 & si\ \{i,j\} \in E \\ 1 & en\,otro\,caso \end{cases}$$

Sea F: $E_{CICLO\text{-}HAM} \rightarrow E_{TSP}$ definida por F(G) =(C,d,0).

- F es total computable en tiempo polinomial.
- Aserto: Para cada G ∈ $E_{CICLO\text{-}HAM}$, G posee un ciclo hamiltoniano sii existe una permutación de {1, ..., n} tal que $\sum_{i=1}^{n} d(\pi(i),\pi(i+1)) \leqslant 0$ (n = | G |).

- Sea γ=(u_1, …, u_n,u_1) un ciclo hamiltoniano de G. Sea π la permutación de {1, …, n} definida por $\pi(i) = u_i \ (1 \leqslant i \leqslant n)$.

  Se tiene que $\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant 0$.

- Sea π una permutación de {1, ⋯, n} tal que

  $$\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant 0$$

  Entonces γ=(π(1), ⋯, π(n), π(1)) es un ciclo hamiltoniano de G.

2.4.- APLICACIONES PRÁCTICAS DEL TSP.

El problema del viajante de comercio tiene un número elevado de aplicaciones prácticas, desde la más directa y obvia, en problemas de logística, hasta otras en la que el TSP está implicado indirectamente, como subproblema de otro más grande.

- **Logística**

  El TSP tiene como aplicación más inmediata y natural todos los problemas relacionado con el área de la logística. Grandes o pequeñas empresas dedicadas al transporte de mercancías donde los vehículos u otros medios de transporte tengan que realizar una ruta por varias ciudades o puntos. Igualmente, esto ocurre con los servicios de correos o postales. Grandes esfuerzos se han realizado para aplicar el TSP en la recogida urbana de basuras, donde los camiones tienen que se asignados a una determinadas rutas realizando cálculos dependiendo de la capacidad del vehículo de recogida de basuras, del número de contenedores de la ruta, densidad del tráfico y la distancia hasta el vertedero.

  Caso particular es el servicio a domicilio de comida rápida, donde un vehículo sirve varios pedidos a la vez. Es decir, el repartidor antes de salir se lleva varios pedidos para lugares distintos pues se considera esta forma mucho más optimizada que el repartidor salga cada vez para servir un sólo pedido. En esta variante, admite un preprocesamiento donde se pueden obtener mejores resultados si las llamadas son clasificadas según su situación en la ciudad, y de esta forma el algoritmo se va a encontrar que

todos sus pedidos se hayan circunscritos a un mismo barrio o zona.

Sin embargo, vemos como las multinacionales del servicio a domicilio no usan un software que optimice el reparto; esto es debido a que conocen estudios como los que hemos presentado anteriormente donde indican que el ser humano es capaz de encontrar una solución bastante buena en poco tiempo, por lo que la tarea de decidir el camino más corto para repartir la comida es responsabilidad del repartidor o conductor del vehículo.

- Fabricación Asistida por computador.

  El problema del viajante de comercio suele ser usado en las cadena de montajes de las fábricas para que los robots y brazos robotizados minimicen el número de desplazamientos que realizan, por ejemplo, al perforar una plancha metálica. Así mismo, se encuentra presente en los caminos que deben realizar las carretillas que desplazan palets y proveen o retiran de material de la planta procesadora.

- Diseño de circuitos de gran escala de integración.

  Conforme se aumenta la escala de integración, mayor se convierte el problema de la disposición de los elementos electrónicos del circuito y su interconexionado. En este área encontramos muchas restricciones o cotas superiores que hay que establecen en el problema del viajante de comercio, pues, por ejemplo, según la frecuencia de trabajo del circuito existe una longitud máxima de una conexión que está determinada y no puede ser sobrepasada, ya que, se convertiría dicha conexión o hilo en una antena emisora dando lugar a numerosos problemas, interferencias en el funcionamiento del circuito, pérdida de potencia eléctrica -que se iría en la radiación de la antena- o malfuncionamiento de otros circuitos adyacentes al mismo.

- Cristalografía de rayos X.

  La cristalografía de rayos X es una técnica que consiste en hacer pasar un haz de rayos X sobre un cristal de una sustancia que se haya en estudio. Debido a la estructura simétrica de las moléculas del cristal y a la difracción, el haz se escinde en varias direcciones; dando lugar a un conjunto de haces e intensidades distintas. Esta técnica es muy utilizada en la descripción de moléculas de ADN y proteínas.

El papel que juega aquí el problema del viajante de comercio es que es usado para optimizar los perfiles de difracción de un haz al atravesar una determinada sustancia.

- Computación

  En el área de la computación tiene muchas aplicaciones como puede ser la optimización en la ejecución de los procesos que tiene que realizar el planificador de procesos de un sistema operativo para el que tiempo global de cómputo sea el menor. También podemos ver la aplicación de este problema en la optimización de redes de computadoras para que la comunicación entre ellas sea más eficiente; esto tiene alcance desde la misma disposición de los distintos dispositivos de red (antenas, cableado, concentradores, conmutadores, etc) y las computadoras que se conectan a ella; hasta la optimización de los protocolos de comunicación para que un paquete tenga que recorrer un camino más corto dentro de una red telemática.

Además de estas, existen muchas más áreas como la economía donde se usa para la optimización en el cálculo de intereses o rentabilidades en el mercado. En la medicina, donde es usada para optimizar o generar caminos menos dañinos a la hora de realizar un trasplante de órganos y cirugía, pues hay que intentar evitar tocar órganos vitales o hacerlo lo menos posible. En astronomía, donde hay que optimizar el movimiento y recorrido de los telescopios y radiotelescopios en las observaciones del firmamento, debido al que el tiempo de exposición de los cuerpos celestes es limitado y los movimientos de las lentes o antenas son lentos y costosos, por lo que el TSP es usado por la ESA, NASA y otras agencias espaciales para crear un camino mínimo entre los cuerpos celestes objeto de la observación a realizar por el telescopio. Y otras muchas áreas que hemos visto que se hayan dentro de la investigación de este problema.

CAPÍTULO 3. REDES P2P

3.1.- INTRODUCCIÓN

Una red de computadoras es la unión de varias computadoras que a través de un medio de transmisión de datos son interconectadas entre sí con el fin de compartir u ofrecer recursos, información, servicios, etc. Estas redes tienen una estructura cliente/servidor, donde un computador de la red adquiere el rol de servidor y los demás son clientes.

Una red P2P (*del inglés Peer to Peer*), que también puede llamarse de varias formas, como, red entre pares o entre iguales; se define como aquella en donde todos o algunos de los computadores que la forman son cliente y servidor simultáneamente respecto a los demás componentes de la red. Las redes P2P son útiles para varios propósitos, pero están especialmente indicadas para compartir información directa entre sus miembros.

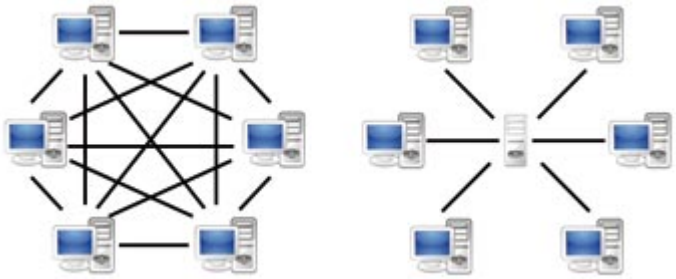


Fig. 3.01. Diagrama de una red P2P y una red de computadores (cliente/servidor)

Las redes P2P normalmente se despliegan en la capa de aplicación de una red de computadores ya existente, habitualmente de uso público como Internet. Respondiendo a una implementación de redes superpuestas.

Una característica principal de las redes P2P es su dinamismo. En estas redes aparecen nodos o desaparecen los mismos de forma no determinista, por lo que no se puede prever su tamaño. De igual manera, los nodos son heterogéneos tanto en sus características computacionales como en su forma de conexión a la red.

3.1.1.- CARACTERÍSTICAS

Las redes P2P cumplen inherentes a su naturaleza las siguientes características.

- **Escalabilidad**. La incorporación de nuevos nodos a la red es muy fácil y no hay una restricción teórica al número de nodos que pueden sumarse a una red P2P. Contrario a lo que podría pensarse, las redes P2P mejoran su funcionamiento cuanto más cantidad de nodos exista en ellas, puesto que, la información estará presente en más lugares y menor es el riesgo de que se pierda al desparecer un elemento de la red.
- **Robustez**. El propio hecho de que una misma información se encuentre replicada en varios o en la mayoría de los nodos de la red, la hace menos vulnerables a caídas de uno o múltiples nodos en la red. Asimismo, la duplicidad de la información en varios nodos elimina los cuellos de botella que se producen en las redes clásicas, puesto que, múltiples peticiones de un mismo recurso pueden ser satisfechas por varios elementos evitando la saturación de un sólo punto de la red.
- **Descentralización.** Por propia definición, los miembros de una red P2P son iguales y/o no tienen funciones especiales. Y, por tanto, ningún nodo es imprescindible en el funcionamiento de la red. Existen algunos tipos de redes P2P, donde la descentralización no es total y pueden existir varios elementos de la red que sí cumplan propósito especiales como proxys, indexadores, etc
- **Capacidad de cálculo**. El equipo servidor en las redes clásicas (cliente/servidor) es el que debe proveer la capacidad para computar todas las peticiones que le requieran los clientes a el conectados. En el caso de las redes P2P esto no es así, ya que, cada nodo suma su capacidad de cómputo a la capacidad de la red total. Así, cuanto más crece la red, mayor es su capacidad de cálculo.

De estas características que hemos mencionado podemos extraer algunas desventajas de las redes P2P como son las siguientes:

- **Desaparición de nodos**. El mismo dinamismo que poseen este tipo de redes y que permiten su alta escalabilidad, pueden producir el hecho contrario: decrecimiento del número de miembros de la red. Hay que

recordar que la aparición o desaparición de nodos es imprevisible y aleatoria.

- **Tolerancia a fallos**. Si bien estas redes presentan una buena tolerancia a fallos debido a la replicación de la información entre los nodos de la red. Puede ocurrir que un descenso generalizado de elementos de la red produzca una pérdida en la información a compartir en la misma. La tolerancia a fallos se verá más comprometida cuanto más descentralizada sea la red P2P.

3.1.2.- ESTRUCTURA DE LA RED

La redes P2P fundamentalmente operan en la capa de aplicación de la torre OSI como redes superpuestas, tal y como hemos comentado, de una red telemática ya establecida. Aún así, la red P2P establece caminos o enlaces entre los nodos que la componen, independientemente de los que exista en la red subyacente, siendo alcanzables estos nodos a través de la red P2P.

El modo en que estos enlaces se realizan en la red superpuesta hace que se pueden clasificar las redes P2P en estructuradas o no estructuradas.

Arquitectura no estructurada

Se habla que la red es no estructurada cuando los enlaces entre nodos de la red superpuesta se establecen de forma arbitraria. Estas redes se construyen muy fácilmente porque cuando un nodo se incorpora se une a otros sin tener que respetar una topología en concreto o que un nodo superior jerárquico le indique que enlaces debe establecer. Incluso durante el tiempo que permanezca en la red los enlaces pueden variar aleatoriamente. Debido a esta circunstancia, estas redes son muy resistente a la incorporación y desaparición de nodos, ya que, no deben mantener en todo momento ninguna topología en concreto y, por tanto, los nodos de la red no tienen que dedicar tiempo a reenlazar con otros. De esta forma, las redes no estructuradas son más resistentes a ataques de denegación de servicio y son capaces de soportar consultas complejas.

El servicio de búsqueda de recursos en estas redes se basa en que la petición de un usuario tiene que recorrer todo o una parte del sistema con la finalidad de encontrar uno o varios nodos que compartan el recurso buscado.

Esta simplicidad en la búsqueda hace surgir dos inconvenientes.

- La búsqueda no siempre se satisface. Los contenidos más populares serán compartidos por un número mayor de miembros de la red y, por tanto, con mayor disponibilidad. Mientras que los recursos menos populares necesitarán de una búsqueda más intensa tanto a nivel de recorrido por la red como de tiempo; llegando, incluso, a no alcanzar dicho contenido (aunque esté presente en la red) porque no exista una camino entre el nodo que hizo la petición y el nodo que lo posee.
- Inundación. Debido a que la búsqueda es sistemática a todos los nodos alcanzables de la red, esto produce un fenómeno de inundación de la red porque se genera mucho tráfico en cada petición. Como consecuencia de ello, el tiempo de obtención de una respuesta aumenta y limita el número de nodos de una red que fácilmente se congestiona. Se puede implementar búsquedas guiadas por heurísticas que mitiguen este efecto de inundación del que adolecen las búsquedas a ciegas.

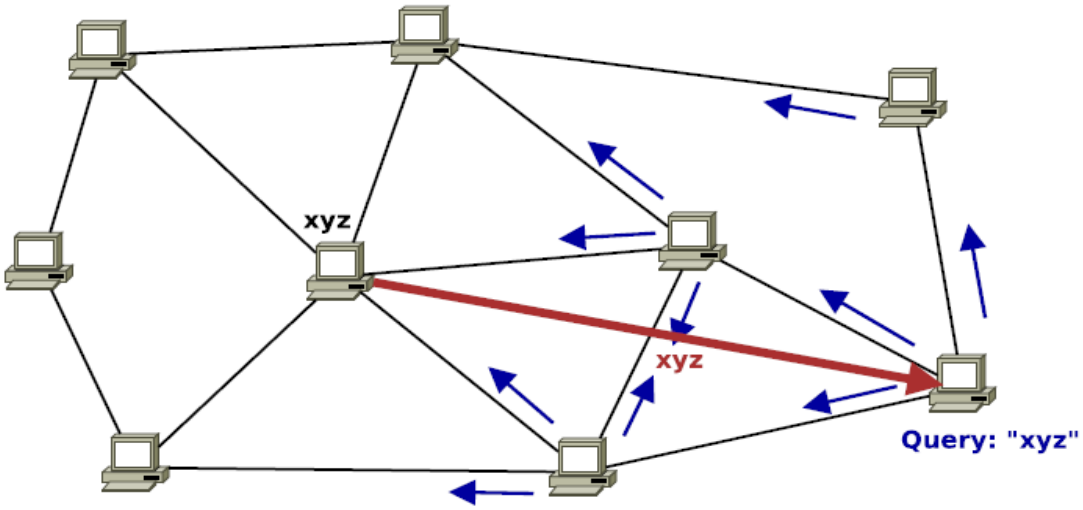


Fig. 3.02. Esquema de una red P2P no estructurada.

La mayoría de las redes P2P más conocidas son no estructuradas, Kazaa, Gnutella.

Arquitecturas estructuradas

Son concebidas para superar los inconvenientes de las no estructuradas. Cada nodo guarda información útil sobre rutas, enlaces y contenidos sobre otros nodos de la red. De esta manera, cada miembro es responsable de una parte específica de la red. Toda esta información es mantenida en una tabla de dispersión *(hash)* distribuida. En ellas, se asigna valores a cada recurso compartido y a cada nodo de la red. Luego cada nodo sigue un protocolo global

de actuación que relaciona los recursos compartidos con los nodos responsables de los mismos.

Así pues, siempre que un usuario desee realizar una búsqueda de un contenido, ha de utilizar el protocolo de actuación para determinar que nodo es el responsable de dicho contenido. Una vez obtenida esta información, tan sólo tiene que dirigir la búsqueda a tal nodo (o varios nodos) que le suministrará el recurso solicitado.

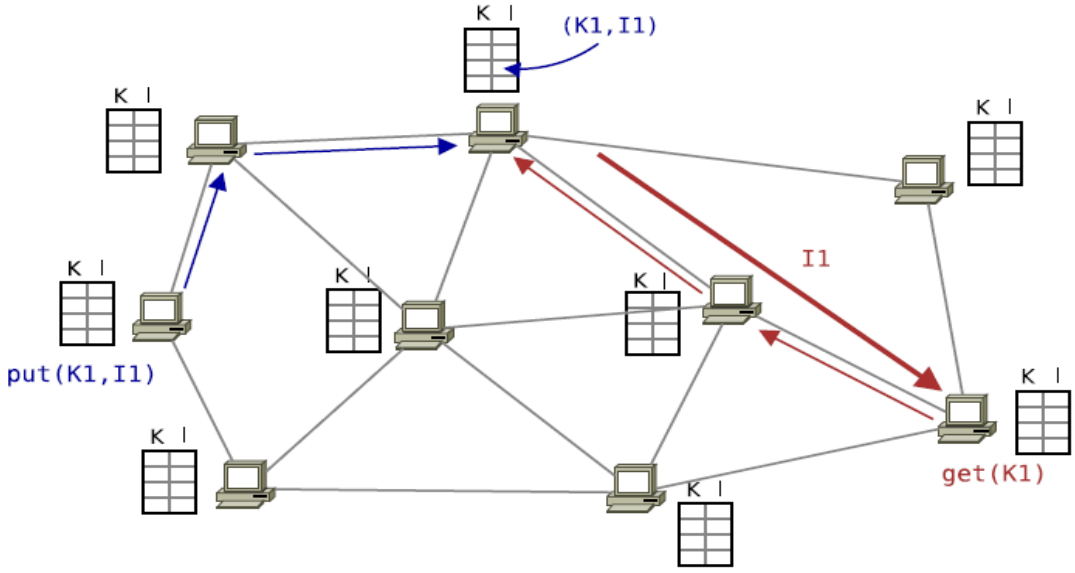


Fig. 3.03. Esquema de una red P2P estructurada. Cada nodo contiene una tabla de dispersión donde se asocia un identificativo del recurso a compartir K, con una dirección del nodo propietario del recurso I.

Cada nodo que se incorpore a la red, debe crear su tabla de dispersión distribuida y cumplir el protocolo global de actuación y mantener una topología de red determinada.

Ejemplo de protocolos usados en redes estructuradas son Chord, Patry, CAN, Kadmelia.

Según se organicen este tipo de redes, se pueden clasificar de la siguiente manera:

- Centralizado. Responden a una estructura cliente/servidor. Existe un servidor central que recopila información sobre los miembros conectados a la red y la información que comparte cada uno de los nodos. Cuando un nodo hace una petición de un contenido, se la hace al servidor. Éste le contesta con la dirección del nodo que posee el contenido solicitado. En ese momento, el usuario lanza una petición al nodo que le ha dicho el servidor y que posee el recurso que busca, para que se le permita acceder

a el. En ningún momento, el servidor almacena información. Ya no existen redes P2P de esta forma, y podemos encontrar ejemplos en la originarias: Napster y Audiogalaxy.

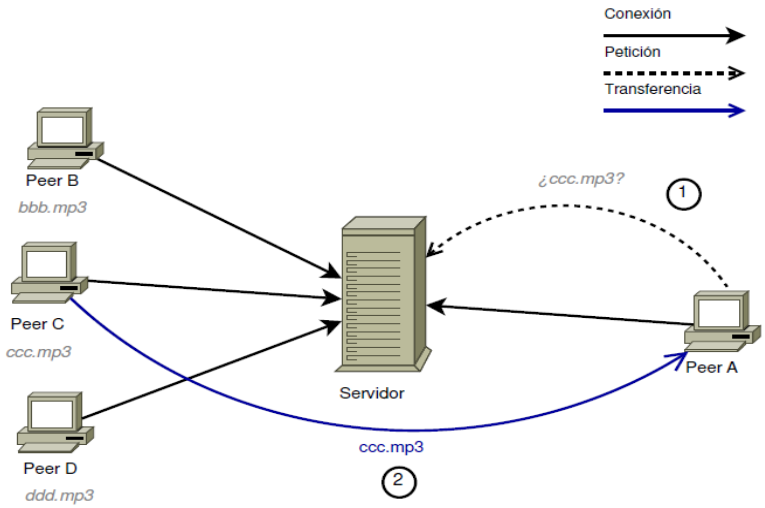


Fig. 3.04. Esquema de red centralizada

- Descentralizado. Este modelo representa la definición de redes P2P, por lo que, suelen llamarles redes P2P "puras"; puesto que, todos los nodos de la red tienen los mismos roles de cliente y de servidor. La comunicación se efectúa de miembro a miembro sin ningún intermediario. A este tipo responden los protocolos Ares y Freenet.

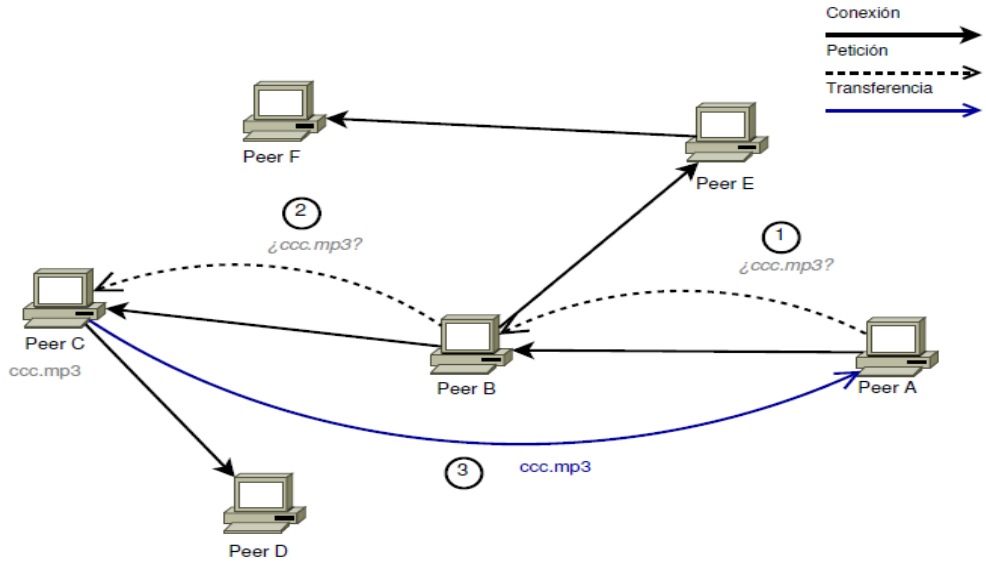


Fig. 3.05. Esquema de red descentralizada

- Mixto o híbridos. En esta caso, algunos nodos de red se encargan de gestionar el tráfico generado por otros nodos, son llamados *superpeers*. Estos superpeers son seleccionados entre los miembros de la red por sus capacidades computacionales y de ancho de banda. Cada nodo gestiona

un determinado número de conexiones, donde a cada una de ellas se le asocia un superpeer. Cuando un nodo se conecta a la red envía una lista de los recursos que va a compartir a su superpeer. Así una solicitud de búsqueda se dirige al superpeer apropiado. Luego éste superpeer puede o no extender el mensaje a otros superpeers. Su funcionamiento es comparable al tipo centralizado pero este modelo es mucho más escalable y tolerante a fallos.

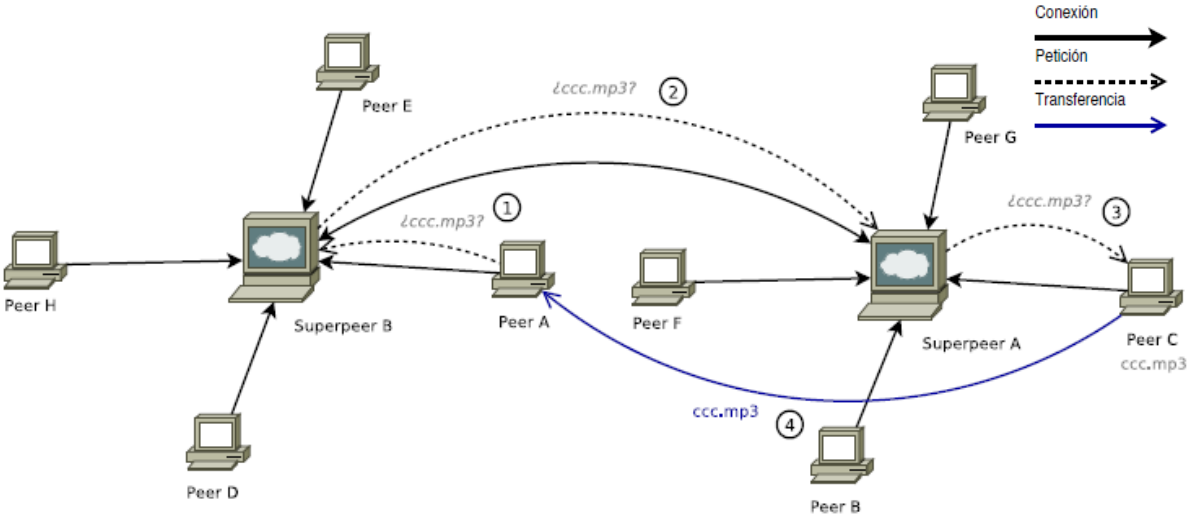


Fig. 3.06. Esquema de red híbrida.

Encontramos redes de este tipo con los protocolos Bittorrent y edonkey2000.

3.2.- COMPUTACIÓN DISTRIBUIDA

El avance de las Ciencias de la Computación en sus distintas ramas, ha permitido que otras áreas del saber planteen nuevas necesidades que de otra manera no podían desarrollar por ser intratables sin la ayuda de la computación. Esta sinergia se retroalimenta cuando se exige más altas capacidades de cálculo desde la ciencia (climatología, medicina, física), la ingeniería (materiales, energía) o la industria (farmacéutica, química, automovilística); y las ciencias de la computación quiere responder a esta demanda.

Históricamente la informática ha tratado de resolver este problema de la potencia de cálculo diseñando *supercomputadores*. Estos sistemas, tanto monolíticos como paralelos, tienen un elevado coste, mantenimiento y son pocos flexibles. Y para algunas aplicaciones se muestran insuficientes.

En el afán de superar estas limitaciones se ha ideado un nuevo modelo,

llamado computación distribuida.

La computación distribuida la podemos definir como el conjunto de computadores independientes, interconectados por una red telemática y que son capaces de colaborar para realizar una tarea o compartir recursos. Como ventajas de la misma podemos mencionar su escalabilidad, concurrencia y tolerancia a fallos; mientras que encontramos inconvenientes como múltiples puntos de fallo y seguridad.

De la aplicación de este modelo han surgido las siguientes estructuras de cómputo:

- Clúster. Consisten en un conjunto de computadoras homogéneas, habitualmente supercomputadores, conectadas entre sí por una red de alta velocidad. En general, los componentes de un clúster se encuentran localizados físicamente en un mismo recinto.
- Grid. Es una nueva técnica en la cual todos los recursos de un número indeterminado de computadoras son englobados para ser tratados como un único superordenador de manera transparente. Los componentes de un grid (recursos, computadoras, clúster, …) pueden estar ubicados remotamente unos de otros, por lo que, suelen hacer uso de redes de área extensa (WAN).
- Computación Peer to Peer. Es el uso de una red P2P para estas tareas mencionadas de cómputo masivo. Son sistemas donde sus miembros son heterogéneos y conectados a una red de área extensa puesto que estarán en distantes localizaciones entre ellos.
  A priori, puede parecer que un grid y la computación P2P son el mismo caso. Difieren en que un grid es un entorno dedicado muy complejo formado por infraestructuras de alto coste adquiridas por grandes organizaciones, cuya administración requiere de software de control sofisticado y técnicos altamente cualificados; mientras una red P2P es simple, compuesta por los recursos que aportan sus usuarios y únicamente bajo el control de la propia aplicación P2P.

3.2.1.- COMPUTACIÓN VOLUNTARIA

La computación voluntaria es un tipo de computación distribuida en la cual los ciudadanos (voluntarios) donan parte de sus recursos informáticos (como la energía que consume su equipo, la capacidad de cálculo de su procesador o el

espacio de sus unidades de almacenamiento) a uno o varios proyectos que requieren de un cómputo masivo. Los proyectos suelen ser de tipo académico (desarrollados por universidades u organismos de investigación) y orientados a la investigación científica.

Puesto que los recursos voluntarios son de propiedad común y manejados por personas no especializadas, el software debe ser simple de instalar y no puede pedir versiones o configuraciones específicas de sistemas operativos.

Este nuevo paradigma de computación no supone únicamente un nuevo recurso de cálculo sino también un nuevo escenario para la divulgación científica que abre un nuevo escenario para el fomento de la cultura científica. Asimismo impulsa a los científicos a dar a conocer sus investigaciones en términos accesibles a fin de sumar voluntarios a sus proyectos.

La computación voluntaria se dio a conocer internacionalmente con el proyecto SETI@Home, en el que los ordenadores de los voluntarios procesan información capturada por el radiotelescopio de Arecibo (Puerto Rico) en busca de señales de vida extraterrestre inteligente. SETI no fue el primer proyecto de computación voluntaria (este fue, GIMPS - Great Internet Mersenne Prime Search) pero sí el más popular.

Tiempo después, la idea inicial de SETI@Home evolucionó y se convirtió en BOINC (Berkeley Open Infrastructure for Network Computing), un software más general para permitir ejecutar múltiples proyectos de computación distribuida en los ordenadores de casa, a elección del usuario. BOINC funciona de forma muy similar a la del SETI@Home original: se instala en cualquier ordenador y entra en acción solamente cuando no se está realizando ninguna tarea, aunque se puede configurar para que utilice un porcentaje de la capacidad de proceso de forma continuada. Una vez que se ha instalado, se puede elegir a qué proyectos se quiere destinar el tiempo de procesamiento. Si se elige más de uno, el tiempo se divide equitativamente entre todos. El procesamiento se interrumpe cuando el ordenador vuelve a utilizarse, y se retoma la tarea desde el punto en el que se dejó cuando el sistema a quedar libre. La plataforma da soporte a 58 proyectos de investigación y cuenta con 585.486 ordenadores activos, 1.836.178 usuarios registrados de 267 países.

Existen multitud de otros proyectos, en España tenemos iniciativas como IberCivis que cuenta con 10.796 voluntarios registrados de 108 países y 7 proyectos de investigación (entre ellos, uno portugués y 4 de centros del CSIC).

La computación voluntaria se basa en el tipo de computación peer to peer que mediante la red Internet, representa una alternativa de bajo coste para que estos proyectos científicos puedan disponer de una plataforma de computación distribuida de alta capacidad. Y en este sentido, se adscribe este trabajo fin de grado.

3.3.- PROTOCOLO NEWSCAST

En la introducción de este capítulo hemos visto la importancia que tiene para una red P2P la estructura que tiene. Según el propósito al que destinemos la red de pares, depende de la correcta elección de dicha arquitectura entre las existentes, se determinará el buen desempeño de la misma.

En el caso concreto de este trabajo fin de grado, donde el requerimiento para la red P2P será una arquitectura no estructurada donde realizar una computación distribuida de problemas computacionalmente complejos (NP-arduo) llevado a cabo por un gran número de nodos, lo cuales deberán ser alcanzables por la red con una tasa elevada y tener un éxito elevado a la hora de realizar búsquedas aleatorias para compartir y distribuir información. Se ha optado por elegir un protocolo, frecuentemente usado en aplicaciones P2P científicas, llamado newscast.

Newscast es un protocolo de los llamados epidémicos. Está pensado para grandes sistemas distribuidos. Newscast tiene como principal ventaja su alta escalabilidad y robustez, ya que no precisa de uno o varios elementos distinguidos -por ejemplo, supernodos- que se ocupen de la estructura de la red ni sobrecargar a los nodos en complejas operaciones de mantenimiento de la topología en las operaciones de inserción o eliminación de nodos. En este protocolo, cada miembro de la red realiza unas sencillas operaciones para mantener la estructura de la red y dispersar la información. Estas operaciones son extremadamente independientes de cada nodo, por lo que, no hay sobrecarga de la red para su mantenimiento, ni tablas de índices centrales que actualizar, etc. Así, si se produce un fallo en un nodo, este no se propaga al resto de la red o su incidencia es mínima. En definitiva, el propio protocolo implementa eficientemente dos servicios básicos que son el talón de Aquiles de los grandes sistemas distribuidos: la gestión del vecindario (estructura de la red) y difusión de la información.

Funcionamiento

Los dos bloques principales sobre los que se construye newscast es un conjunto de agentes y una agencia de noticias. La agencia de noticias pregunta periódicamente a los agentes si tienen una noticia y, además, les provee de noticias de los otros agentes del conjunto.

De esta manera los agentes están despreocupado de la red o su entorno y se encargan de desarrollar su función (escucha de sensores, procesar información, cálculos, …) que es generar noticias que irán acumulando hasta que la agencia les pregunte por ellas.

Para un agente un elemento del tipo noticia tiene los siguientes campos:

- *timestamp*: Sello de tiempo. Momento de la creación de la noticia.
- *content:* Contenido de la noticia. Puede ser vacío.

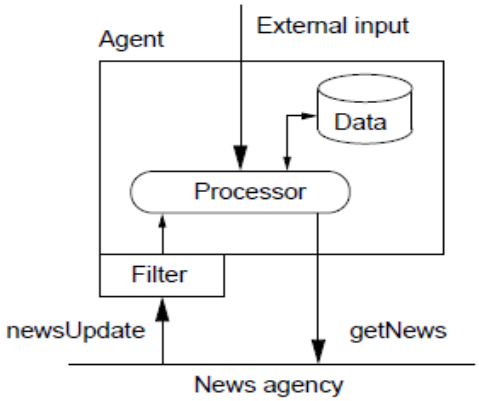


Fig. 3.07. Organización de un agente

Como los agentes son pasivos en las tareas de comunicación y nunca la inician, es la agencia la que posibilita dicha comunicación. Para ello, todos los agentes deben implementar las siguientes operaciones, g*etNews()* y *newsUpdate(news[]).* La agencia de noticias será quien las llames de forma periódica. Sin entrar en detalles de la implementación la función *getNews()* devuelve un item noticia si se ha generado o devuelve vacío. El método *newsUpdate(news[])* le pasa al agente las noticias que han proporcionado el resto de agentes del conjunto.

Damos por sentado que cada agente está ubicado en cada nodo de la red. Pero que la agencia de noticias es un ente abstracto y común a toda la red. Pues bien, cada agente tiene asignado un corresponsal en concreto para el, y es el representante de la agencia desde el punto de vista del agente. Este corresponsal también se ejecuta en el mismo nodo que el agente que le corresponde.

Cada corresponsal contiene un caché local de elementos del tipo noticias, esta caché tiene un tamaño fijo que es determinado al crearse la red P2P. Hay que mencionar que el tamaño de la caché se corresponde con la longitud del array *news[]* que se pasa como parámetro a la función *newsUpdate(news[])*.

Para un corresponsal un elemento del tipo noticia tiene los siguientes campos:

- *timestamp*: Sello de tiempo. Momento de la creación de la noticia.
- *content:* Contenido de la noticia. Puede ser vacío.
- *agentID:* identificador único del agente que ha generado la noticia
- *address:* dirección de red del corresponsal del agente identificado por ID

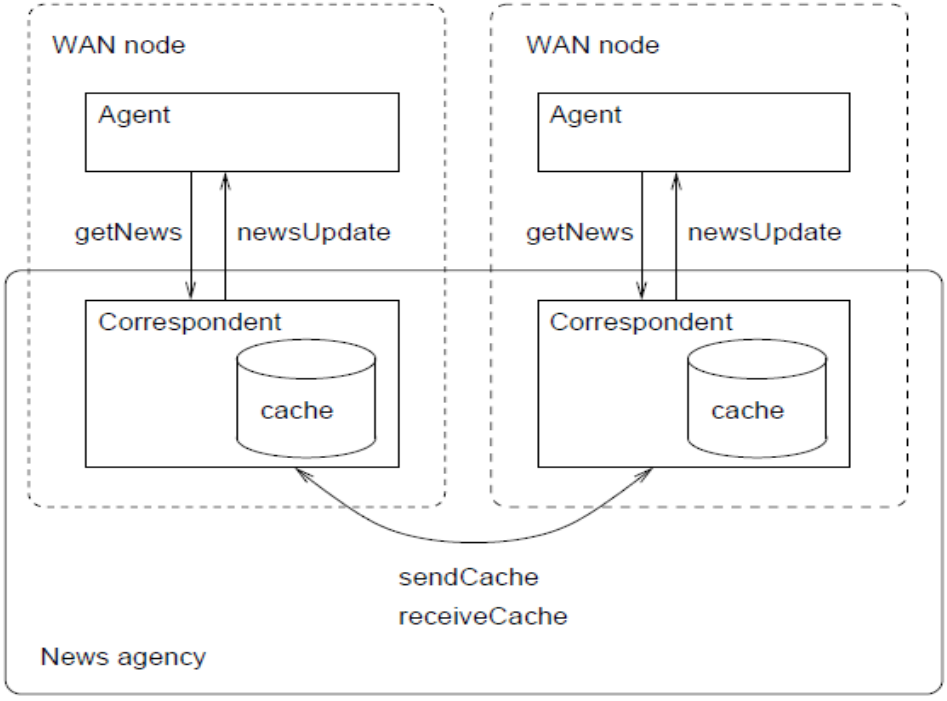


Fig. 3.08. Arquitectura de newscast para dos agentes

Los corresponsales se intercambiar regularmente las cachés con el fin de diseminar en la red la información. El proceso ocurre de la siguiente manera:

1. La agencia de noticias, por mano de su corresponsal, le solicita una noticia al agente - *getNews()* -. Esa noticia se incorpora a la caché.
2. Se selecciona aleatoriamente una dirección - campo *address* -, de las almacenadas en la caché, que corresponda a otro corresponsal. Es decir, se elige un nodo vecino.

3. Se envía todo el contenido de la caché al nodo vecino seleccionado. Simultáneamente se recibe la caché de nodo remoto. La nueva información recibida y la preexistente en la caché se mezclan.
4. La nueva información recibida por el nodo vecino se envía al agente por su corresponsal, a través de la llamada del método *newsUpdate(news[])*.
5. Los corresponsales de ambos nodos tienen ahora el doble de elementos del tipo noticia. Posiblemente, la caché no pueda albergarlos a todos. En este caso, el corresponsal desechará las entradas más antiguas, quedándose con la información más actual.

Al deshacerse de las entradas más antiguas, estamos realizando dos tareas muy importante. Desaparece información antigua de la red que viene en los campos *content* del elemento noticia y, además, desaparece direcciones de nodos antiguos que pudieran ser que estén inactivos, campo *address*. Por este motivo, habíamos hablado que el protocolo newscast era capaz de gestionar la estructura de la red y dispersar la información eficientemente y sin un costo computacional extra para la realización de estas tareas.

3.4.- EL SIMULADOR DE REDES P2P PEERSIM

Como en el caso de este trabajo fin de grado, existen circunstancias en las que es necesario contar con un red de pares (P2P), sin embargo, tener un entorno real es bastante caro y difícil de reproducir. En estos casos, existen varios programas que simulan una red de este tipo.

De entre los existentes se ha elegido Peersim debido a que es un software de fuentes abiertas bajo licencia LGPL desarrollado como parte del proyecto BISON (http://bison-project.eu), por el Departamento de Ciencias de la Computación de la Universidad de Bolonia (Italia), basado en el lenguaje de programación JAVA y posee un framework con el que se puede desarrollar toda clase de aplicaciones para redes P2P no estructuradas. Podemos destacar las siguientes características:

- Capacidad para simular redes de más de un millón de nodos.
- Gran dinamismo. Soporta la entrada y salida de nodos de la red de manera continua.
- Dos modos de funcionamiento. Motor de ciclos simplificado (*cycle-based*

engine) y otro motor basado en sucesos (*event-based engine*).

- Se pueden implementar componentes para recopilar datos estadísticos.

Como desventaja de PeerSim es que la red TCP/IP subyacente no está modelada. En el caso de este trabajo fin de grado no afecta esta circunstancia, ya que no es de interés y sólo necesitamos acceder a la red superpuesta.

A la hora de desarrollar un prototipo para el simulador PeerSim hay que tener en cuenta lo siguiente, pues a priori el lector puede hacerse una idea equivocada del funcionamiento del simulador. Los protocolos desarrollados tienen que ser específicamente implementados mediante el framework de Peersim. Esto es, todos los objetos creado durante la simulación son instancias de clases que implementan una o más interfaces de Java o heredan una clase que forma parte de la simulación. En definitiva, el código creado únicamente puede ser ejecutado desde el entorno de PeerSim y no puede ser directamente trasladado a un entorno real; es necesario realizar algunas adaptaciones para ello.

3.4.1.- MODOS DE SIMULACIÓN

PeerSim tiene dos formas de realizar las simulaciones pues implementa dos modelos de planificación de la simulación que ya hemos mencionado con anterioridad.

- Motor basado en ciclos. Este modelo es muy simple con el fin de alcanzar una máxima escalabilidad y rendimiento. El precio que se paga es una pérdida de realismo, en cuanto se carece de la capa de transporte y concurrencia; esto quiere decir que los nodos se comunican directamente entre ellos y que el simulador da el control a los nodos de manera secuencial respectivamente. Este motor está especializado en protocolos epidémicos y está probado para un tamaño de la red mayor a 10^7 nodos. Este modelo basado en ciclos está implementado por la clase *CDSimulator* del paquete *peersim.cdsim.*
- Motor basado en sucesos. En este modelo el planificador del simulador en lugar de gestionar ciclos, gestiona los sucesos o mensajes que van generando los distintos protocolos. Los protocolos de este motor pueden enviar y recibir mensajes de otros protocolos. La implementación en PeerSim se basa en la clase *EDSimulator* del paquete *peersim.edsim*. Este modelo es capaz de simular la capa de transporte para permitir la gestión

de mensajes. Por tanto, añade mayor realismo a la simulación. Este motor puede ser usado en protocolos epidémicos y normales también, además, es capaz de ejecutar los protocolos desarrollados para el motor de ciclos. Se ha probado para un tamaño de la red de 2,5 x 10^5 nodos.

En el caso de este trabajo fin de grado, usaremos el motor basado en ciclos puesto que para el propósito del mismo no es necesario acceder a detalles de la P2P como su capa de transporte y sí necesitamos una buena gestión de un protocolo epidémico.

Así, lo que, se describa de PeerSim en los apartados siguientes estarán basado en el modelo basado en ciclos.

3.4.2.- COMPONENTES DE UNA SIMULACIÓN

Como bien hemos dicho, todo componente que formará parte de un prototipo o experimento en el simulador PeerSim, se tratará de objetos que hereden alguna clase o implementen algunas interfaces.

- Network (Red). Es una clase que alberga una estructura de memoria global a la simulación que contiene todos los nodos de la red. Representa a la red superpuesta que componen una red P2P. Esta clase provee de los métodos básicos para la gestión de la misma y proporcionar información de su estado.
- Node (Nodo). Es una interfaz que describe a un nodo de la red. Proporciona una dirección de red (*address*) para cada nodo creado. Cada nodo de la red tiene un número determinado de protocolos que definen su acción en la simulación. Este número es común a todos los nodos de la red. Así que, este clase tiene que referenciar a los protocolos que ejecuta el nodo.
- Protocol (Protocolo). Un protocolo define una acción que el nodo tiene que realizar durante la simulación. Generalmente una nodo contiene a varios protocolos, ya que, debe hacer varias cosas: gestionar su vecindario, comunicarse con otros nodos, etc. Cada protocolo está identificado por un número llamado *Pid*.
- Linkable. Esta interfaz gestiona la topología de la red desde el punto de vista del nodo. Proporciona a los nodos la forma de acceder y como se conecta con otros nodos de la red. Además, da información de su estado y propiedades. Por ejemplo, el protocolo newscast tiene que implementar

esta interfaz.

- Control. Es una interfaz que tienen que implementar varios tipo de procesos complementarios a la simulación, tales como los *inicializadores* que determinan el estado inicial de los distintos componentes de la simulación -tienen la particularidad que son ejecutados una sola vez durante el tiempo de la simulación-, los *controladores* que modifican las propiedades de la simulación o de alguno de sus componentes o establecen la condición de parada según algún criterio, como por ejemplo, se ha obtenido el resultado esperado y, por último, los *observadores* que recopilan información durante el tiempo de la simulación para generar estadísticas, análisis de la información generada hasta el momento, etc

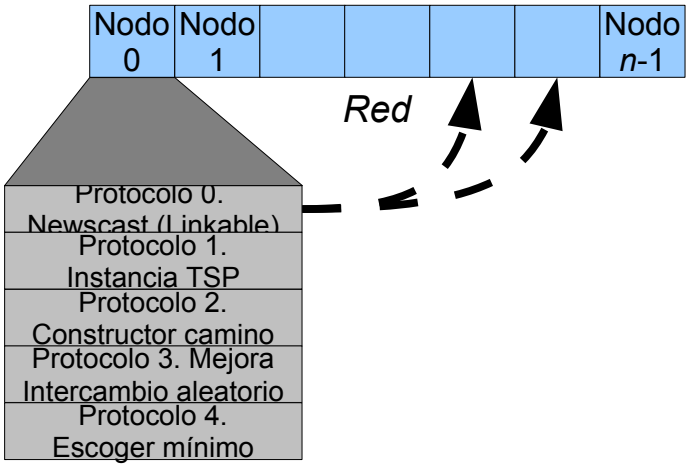


Fig. 3.09. Esquema de red de un prototipo de Peersim

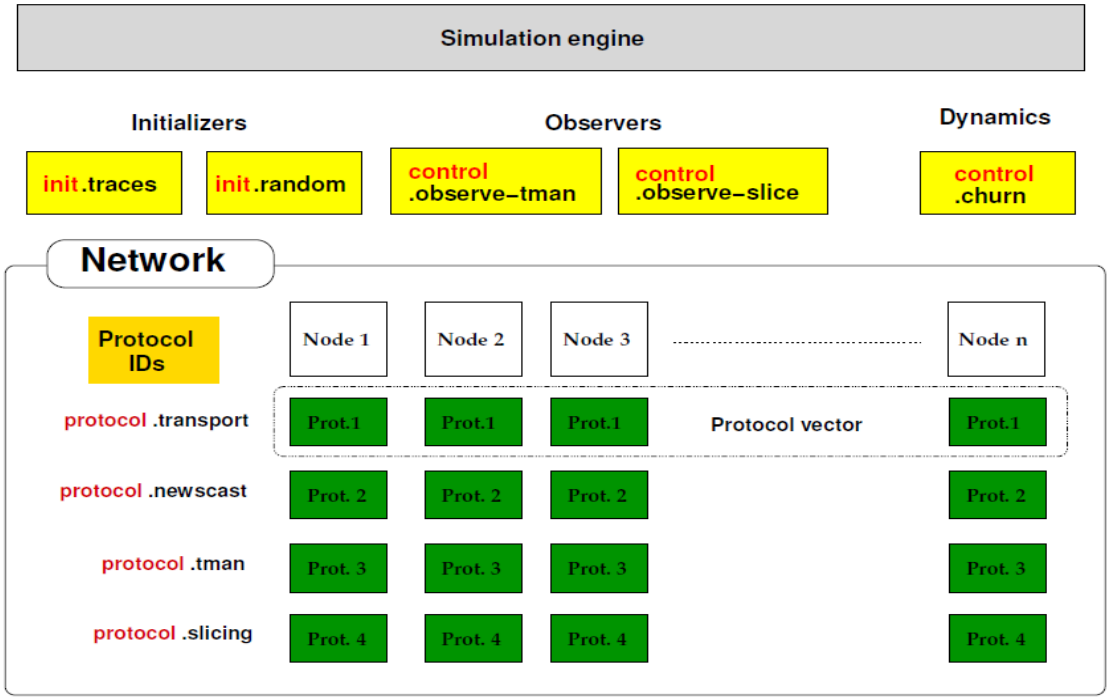


Fig. 3.10. Arquitectura del simulador Peersim

- Dynamics. En realidad, este no es un componente separado de Peersim. Pertenecen a la interfaz Control y está englobado dentro del tipo de los *controladores*, porque modifican las propiedades de la simulación. Lo ponemos aparte por su importancia con una de las características de la redes P2P, que es su dinamismo. Esto es, la entrada y salida continua de nodos en la red. Este componente tiene que simular dicho dinamismo y puede hacerse de muchas maneras, desde una simple implementación aleatoria a modelar un comportamiento particular en grandes sistemas, como redes libre de escala.
- NodeInitializer (Inicializador de nodos). Este componente, que es una interfaz, está unido al dinamismo de una red P2P. En concreto, al suceso de la incorporación de un nuevo nodo durante el ciclo de vida de la red (o la simulación). Cuando un nuevo nodo tiene la intención de unirse a una red ya en funcionamiento, debe recibir o conocer la forma en la que se va a conectar con otros nodos ya existentes, que recursos o información va a recibir de la red y cuales el nodo va a compartir a la misma. Es lo que se llama la inicialización de un nodo. Es una interfaz de PeerSim que debe implementar todo este proceso descrito, donde existirá una inicialización por cada protocolo que contenga el nodo.
- Fichero de configuración. Una vez se implementado y codificado el comportamiento de un experimento para una simulación en Peersim, es decir, todos los componentes anteriormente descritos en este mismo apartado. El simulador ha de recibir una serie de parámetros globales de propiedades de la red y la propia simulación (tamaño de la red, tiempo de la simulación, etc), los protocolos y otros componentes que formarán parte de la simulación. Toda esta información se describe en un fichero de configuración, que el simulador leerá y seguirá a modo de guión de como ha de ejecutarse la simulación.

3.4.3.- CICLO DE VIDA DE UNA SIMULACIÓN

PeerSim está diseñado para fomentar la programación modular basada en objetos, construyendo bloques. Cada bloque es fácilmente reemplazable por otro componente que implemente la misma interfaz. La idea general del modelo de simulación es el siguiente:

- Escoger un tamaño de la red.

- Escoger uno o más protocolos con los que experimentar, tras inicializarlos.
- Escoger unos o más objetos de tipo *Control* para monitorizar las propiedades que interesen y/o modificar algunos parámetros durante la simulación (tamaño de la red, estado interno de los protocolos, etc).
- Ejecutar la simulación invocando la clase *Simulator* con un fichero de configuración.

Todos los objetos creados durante la simulación son instancias de clases que implementan una o más interfaces.

El ciclo de vida de una simulación con el motor basado en ciclos es el siguiente:

1. Lectura del fichero de configuración, dado como parámetro al ejecutar el simulador en la línea de comandos.
2. El simulador establece la inicialización de los nodos de la red, y de sus respectivos protocolos. Cada nodo tiene la misma clase y número de protocolos. Las instancias de los nodos y de los protocolos son creadas por clonación. Esto es, sólo una instancia aparece por la llamada a su constructor de clase -normalmente al formarse el primer nodo de la red-, el cual sirve como prototipo al resto de nodos de la red que son clones de este mismo. Por este motivo, es muy importante prestarle atención en la implementación de los protocolos a los métodos de clonación.
3. Se procede a la inicialización de la red recién creada. Los nodos y protocolos son llevados a su estado inicial. Este proceso es realizado por objetos de tipo *Control* que son planificados una sola vez al inicio de cada experimento. En el fichero de configuración son reconocibles estos objetos porque llevan el prefijo *init*.
4. El motor de ciclos programa el planificador para que se llame a todos los componentes (protocolos y controles) una vez por ciclo en cada uno de los nodos. Esto se repite todos los ciclos que dure la simulación. El final del experimento se dará porque se han consumido todos los ciclos que se dieron al inicio o porque algún *controlador* ha decidido pararlo.

   Cuando un *observador* (objeto de tipo *Control*) tiene que recopilar datos para la elaboración de estadísticas, estos son redirigidos a la salida estándar o volcados en un fichero.

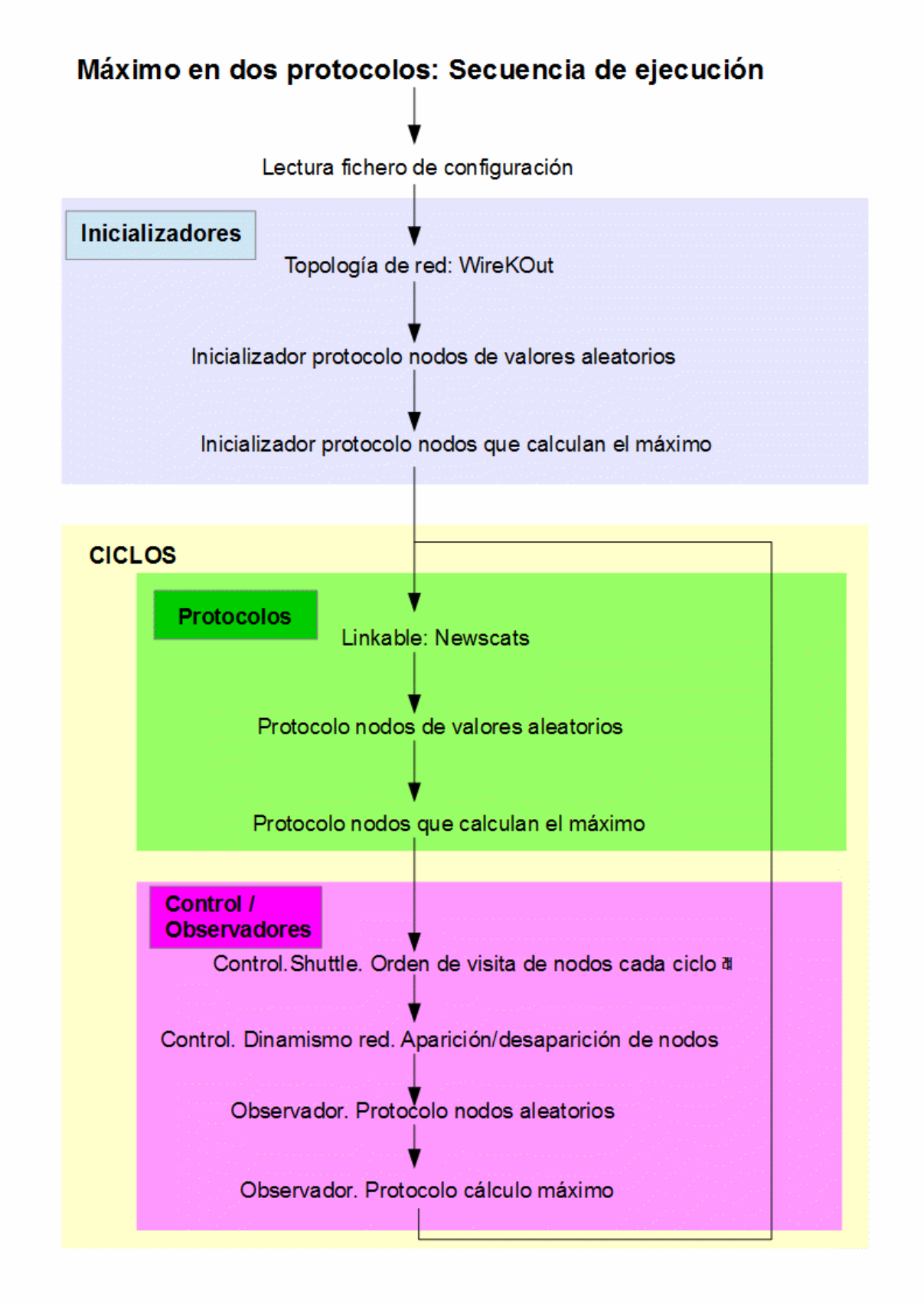


Fig. 3.11. Diagrama que muestra el flujo de ejecución de una simulación muy simple. El objetivo del experimento es encontrar el número máximo de una red donde cada nodo va

generando número aleatorios. La red se inicializa estableciendo una topología inicial (WireKOut) para el protocolo Linkable y, también se inicializa, el resto de protocolos. Cada nodo, además, de Linkable, tiene dos protocolos más que van encaminados al objetivo del experimento. Uno genera en cada ciclo número aleatorios. El siguiente se queda con su máximo local, comunica a otros el suyo u obtiene un máximo de un vecino.

3.4.4.- EL FICHERO DE CONFIGURACIÓN

El fichero de configuración es un fichero en texto plano codificado en ASCII, compuesto por parejas clave-valor, el cual será procesado en Java por la clase *java.util.Properties*; las líneas que comienzan por el carácter # se consideran comentarios y son ignoradas.

La creación del fichero de configuración es el último paso en el desarrollo de una simulación en PeerSim. Esto es, previamente se ha modelado todo el comportamiento en los protocolos, definido la topología de la red e implementado todos los objetos *Control*.

Una vez realizado todo esto, tan sólo, queda definir un guión para PeerSim que le comunique los parámetros del entorno de la simulación y de los objetos tiene que ejecutar y su orden. Es este guión el fichero de configuración.

El fichero de configuración se especifica como parámetro en la llamada a PeerSim para iniciar la simulación en la línea de comandos de la siguiente manera:

```
java peersim.Simulator config-file.txt
```

El contenido del fichero de configuración se divide en cuatro partes principales:

- Parámetros globales
- Declaración y configuración de los inicializadores
- Declaración y configuración de los protocolos
- Declaración y configuración de los objetos *Control.*

Cuya sintaxis general, excepto para los parámetros globales, es la siguiente:

```
<protocol|init|control>.string_id [full_path_] classname
```

y en caso de que la clase contenga parámetros de configuración, la sintaxis será:

```
<protocol|init|contro>.string_id.parameter_name parameter_value
```

Veamos un ejemplo sobre el fichero de configuración del experimento del

máximo referido en la figura 3.11 que clarificará lo expuesto.

```
# CONFIGURACION DE MÁXIMO

MAXVAL 24000

random.seed 1234567890
simulation.cycles 30

network.size 15000
network.node peersim.core.GeneralNode

order.protocol lnk,rndnodes,max
order.control shf,dnet,rndo,maxo

debug.config

# Linkable initializer
init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k 10

# Protocols initializer
init.rninit RandNodesInitializer
init.rninit.value MAXVAL
init.rninit.protocol rndnodes

init.maxinit InitMaxPeers
init.maxinit.value MAXVAL
init.maxinit.protocol max

# Protocols

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache 10

protocol.rndnodes RandNodes
protocol.rndnodes.value MAXVAL

protocol.max MaxInPeers
protocol.max.protocol rndnodes
protocol.max.linkable lnk
```

Constante y su valor

Parámetros globales

Le decimos al planificador el orden de ejecución. Si no se indica, se elige el orden en que son declarados.

Lectura del fichero de configuración en modo depuración

Topología inicial

Objetos Control del tipo inicializadores. Un inicializador por cada protocolo.

Protocolo 1: Newscast. Topología

Protocolo 2: Genera valores aleatorios

Protocolo 3: Comunica con la red y escoge el máximo

Declaración de los protocolos. Podemos observar que en primer lugar se referencia el nombre de la clase que implementa al protocolo y después se introducen los valores de los parámetros que necesita cada protocolo. Esto es igual en inicializadores y control.

```
# Controller and Observers

# Order of nodes are visited
#control.shf Shuffle

control.dnet OscillatingNetwork
control.dnet.maxsize 15000
control.dnet.minsize 10000
control.dnet.period 3
control.dnet.init.0 StarNI
control.dnet.init.0.protocol lnk
control.dnet.init.1 RandNodesNI
control.dnet.init.1.protocol rndnodes
control.dnet.init.1.value MAXVAL
control.dnet.init.2 MaxInPeersNI
control.dnet.init.2.protocol max

control.rndo RandNodesObserver
control.rndo.protocol rndnodes

control.maxo MaxInPeersObserver
control.maxo.protocol max
control.maxo.value MAXVAL
control.maxo.file true
control.maxo.filename stats.txt
```

Controlador para simular la entrada y salida de nodos. Dinámica de la red.

Objetos para inicializar los nuevos nodos que entran en la red. Uno por cada protocolo.

Observadores que recopilan información y, en su caso, la vuelcan a un fichero

En la documentación del paquete *peersim.config* del PeerSim se detalla todas entradas que acepta el fichero de configuración.

3.4.5.- INICIALIZADORES DE TOPOLOGÍAS DE RED.

PeerSim al ser un proyecto abierto cuenta con un número considerable de aportaciones. Entre ellas nos encontramos varias implementaciones de inicializadores de topologías de red para los experimentos. También podemos nosotros desarrollar uno a nuestra medida y que se ajuste a nuestras necesidades. Como la colección con la que cuenta PeerSim de serie es interesante, vamos a enumeralas en este apartado:

- **WireGraph:** Clase abstracta que implementa la interfaz Control. De ella derivan todas las clases que definen la topología de red. Obliga a tomar un protocolo "linkable".
- **WireKOut:** Añade conexiones aleatorias. Nunca elimina conexiones, siempre añade. No produce ciclos. Toma un parámetro *k* que es el número de conexiones aleatorias que genera.

- **WireFromFile:** Toma un fichero donde se especifica cada nodo de la red con su lista de vecinos. El formato del fichero es el siguiente: un fichero de texto plano donde en cada línea se indica en primer lugar el ID del nodo a conectar. Seguido y separado por espacios en blanco de los ID de los vecinos de dicho nodo. Si el ID del nodo es mayor que el tamaño de la red, se obvia esa línea. Si la línea empieza por el carácter # tampoco se considera. Se puede expresar un parámetro *k* que indica el número de vecinos que se van a tener en cuenta del fichero.
- **WireRegRootedTree:** Forma un árbol donde cada nodo tiene *k* hijos, donde *k* es un parámetro dado. Los hijos de cada nodo son sus vecinos. Así el nodo 0 tiene de vecinos desde el 1 hasta *k*. En general, el nodo *i* tiene de vecinos desde *i*k+1* hasta *i*k+k*
- **WireRingLattice:** Forma un anillo mallado donde cada nodo se une con sus circundantes hasta un grado *k*. *K* es el parámetro que define el grado de la malla. El grado del grafo resultante es *2k.*
- **WireScaleFreeBA:** Forma una red de las llamadas libre de escala mediante el modelo de Barabási–Albert. El método toma el parámetro *k,* que define el número de aristas que se añaden a cada nuevo nodo. El método implementado en Peersim toma inicialmente un conjunto inicial de nodos de tamaño *k*. El primer nodo añadido se conecta a todos los nodos del conjunto inicial y después se aplica el modelo Barabási–Albert.
- **WireScaleFreeDM:** Forma un red libre de escala siguiendo el modelo descrito por S.N. Dorogovtsev, J.F.F. Mendes. Este modelo de red bioinspirado se diferencia del anterior en que advierte que la mayoría de las conexiones suelen ocurrir entre los nodos más cercano. Es decir, la distancia de las aristas son cortas. Los autores se basan en el efecto conocido como "*mundo pequeño*" o "*small-world*". Toma un parámetro *k* que será dos veces el número de arista que se añaden a un nuevo nodo.
- **WireStar:** Forma una estrella con las conexiones de cada nodo. No hay parámetros.
- **WireWS:** Forma un red libre de escala siguiendo el modelo de Watts y Strogatz que también está basado en el efecto de redes de "*mundo pequeño*". La implementación de Peersim tiene un pequeña variación del original, ya que, por defecto toma de entrada un grafo dirigido. Las aristas

no dirigidas son convertidas a un doble enlace, uno por cada sentido.

Redes libre de escala

Muchas de las topologías, que hemos mencionado, hacen referencia a redes libre de escala. Esta referencia no sólo ocurre en estos inicializadores sino que en este trabajo fin de grado, se puede ver en inicializadores de nuevos nodos para redes dinámicas que han sido desarrollados para este trabajo; y en el propio protocolo de newscast.

Hemos usado estos objetos de PeerSim como justificación para introducir estas redes que mantienen una estrecha relación con las redes de pares.

Las redes libre de escala son un tipo de redes complejas. Especialmente, la mayoría de redes complejas de gran tamaño son de este tipo. Como característica que las defines podemos decir que en las redes libre de escala algunos nodos están altamente conectados, es decir, poseen un gran número de enlaces a otros nodos. Sin embargo, el grado de conexión de casi todos los nodos es bastante bajo. A esto, podemos unir, que muchas de ellas poseen un fuerte dinamismo.

Sin embargo, la mayoría de las redes libres de escala son muy sensibles a eliminaciones dirigidas (no aleatorias); basta eliminar una proporción muy baja de nodos específicos (por ejemplo, aquellos de grado alto, los *concentradores*) para que la red quede completamente dividida y muy dispersa.

Este último hecho indica que la red es muy frágil. Esta fragilidad tiene sus raíz en la naturaleza extremadamente no homogénea de la distribución de grados de las redes libres de escala.

Las redes P2P casan perfectamente con este modelo, puesto que, suelen tratarse de redes de gran tamaño donde son fácilmente localizables los recursos más populares (alto grado de conexiones) mientras que el resto de información compartida en la red P2P no se encuentra masivamente enlazados.

La teoría de estas redes surge en 1999 cuando László Barabási y sus colaboradores de la Universidad de Notre Dame en Indiana (EE. UU.) hicieron un mapa de la Web en 1999. Para su sorpresa, la web no presentaba una distribución del grado de conectividad usual. En lugar de esto, algunos pocos nodos, a los que llamaron *concentradores*, estaban mucho más conectados que el resto. Este fenómento también fue observado por otros investigadores en fechas cercanas. Por lo que, dio origen al modelado y estudio de estas redes que están presentes

en diversos aspecto de la biología y sociología. En consecuencia, es un campo que actualmente está en plena actividad.

Ejemplos de redes libre de escala, además de las rede P2P o web, pueden ser las siguientes:

- La red de amistades entre personas. Esto también se puede extender a las redes de llamadas telefónicas, de envíos postales y de correo electrónico.
- La red de distribución eléctrica, en que existen estaciones enormes que abastecen a zonas enormes, y al mismo tiempo una miríada de transformadores pequeños.
- Las redes de comercio internacional, dado que los países desarrollados, que son la minoría, concentran la mayor cantidad de intercambio de bienes, mientras que en los países no desarrollados, que son la mayoría, el intercambio comercial es relativamente menor. Esto se aplica también a las redes de comercio entre empresas dentro de cada país.
- Las redes de citaciones bibliográficas incluyen unos pocos libros o escritos muy citados, mientras que la mayoría de los libros reciben pocas o incluso ninguna citación.
- Las redes de neuronas en los organismos dotados de sistema nervioso, lo que significa que permanentemente usamos mucho una fracción de las neuronas, mientras que la mayoría de las neuronas las ocupamos muy poco.
- Las redes de interacción de proteínas en el metabolismo celular, con unas pocas proteínas que aparecen en la mayoría de las reacciones, mientras que la mayoría de las proteínas aparecen sólo en situaciones muy específicas.
- Las redes de caminos, pues la mayoría de los caminos llegan a unas pocas ciudades muy grandes, mientras que de la mayoría de ciudades pequeñas salen unos pocos caminos. Lo mismo es válido para las rutas marítimas y los puertos, las rutas aéreas y los aeropuertos.

CAPÍTULO 4. IMPLEMENTACIÓN DE LOS METAHEURÍSTICOS.

4.1.- DESCRIPCIÓN GLOBAL

El principal objetivo de este trabajo es la implementación de algunos metaheurísticos conocidos para que puedan ser ejecutado de manera distribuida en un entorno de una red P2P. Como sabemos los metaheurísticos son técnicas que pueden ser aplicadas a la resolución de diversos problemas.

Con el fin de que en la parte de análisis se puedan establecer comparaciones más fiables, todas los metaheuríticos que se van a implementar se harán de acuerdo a que resuelvan el mismo tipo de problema: el problema del viajante de comercio. Esta elección se basa en, como hemos mencionado en el capítulo 2, uno de los problemas más estudiados y que es banco de pruebas para casi todas las técnicas heurísticas es precisamente el problema del viajante de comercio.

Desde el punto de vista del problema del viajante de comercio los heurísticos se engloban a los dos grandes grupos, heurísticos de construcción y heurísticos de mejora respectivamente. Por lo que las implementaciones de estos irán dirigidas a esta clasificación.

Recordemos, que los heurísticos de construcción son aquellos algoritmos que a partir de una instancia dada para el problema del viajante de comercio, crean un camino o *tour,* como suele denominarse dentro de la nomenclatura del TSP, cuyo valor ya se encuentra relativamente próximo a un óptimo, al menos local, en comparación a lo que pudiera obtener un método completo en el mismo tiempo de cómputo; es decir, el camino que genera el heurístico de construcción es de un calidad muy aceptable y puede considerarse un buen punto de partida para iniciar una búsqueda, desde otro método (heurístico o completo), para encontrar el camino óptimo o alguno que se le aproxime. La estrategias para crear los caminos son variadas y cada una de ellas constituyen un heurístico en cuestión o una familia de ellos.

Por otro lado, hemos visto que los heurísticos de mejora son métodos que

toman ya un camino o tour construido con anterioridad para una determinada instancia del problema del viajante de comercio, generalmente el camino es generado por un heurístico de construcción, y da como salida otro camino cuyo valor es menor o igual (en el peor de los casos) que el camino dado originalmente. El algoritmo de mejora básicamente realiza operaciones con las aristas del camino de entrada, mayormente eliminación de unas y adición de otras nuevas, donde la aplicación de dichas operaciones están basadas en experiencias y desarrollos que sus autores han observado en sus investigaciones. Obviamente, cuanta más calidad tenga el camino de entrada al método de mejora, mucho mejor será su resultado.

Además de los propios heurísticos, se han implementado otras clases que las hemos incluidos en paquete de Java. Todos los paquetes se listan a continuación:

- Instances. Este paquete provee clases para albergar un instancia del problema del viajante de comercio. Estas instancias pueden ser creadas a partir de valores aleatorios o a través de un fichero de coordenadas en 2 dimensiones escrito en formato para instancias euclidianas utilizado en el repositorio TSPLIB.
- Constructor. Este paquete contiene los heurísticos de construcción. En el hay heurísticos implementados como clases normales de Java y como protocolos de PeerSim.
- Improvers. Análogo al paquete constructor, aquí se encuentran los heurísticos de mejora, algunos implementados como clases o como protocolos.
- Dynamics. Objetos de tipo control para la gestión del dinamismo de la red P2P.
- Vector. Protocolos y objetos control que afectan a todos los nodos de la simulación.
- Utils. Clases y objetos control de ayuda a la otras clases o a la simulación.

4.2.- PAQUETE PARA LA CREACIÓN DE INSTANCIAS.

4.2.1.- LA CLASE DataTSP.

Es la clase padre para la creación general de instancias para el problema del

viajante de comercio. Una instancia del TSP sólo es necesaria crearla una única vez, ya que, generalmente no cambia durante el proceso o, en nuestro caso, simulación. Por este motivo, esta clase implementa la interfaz *Control* porque se usará como inicializadora de un protocolo cuya misión sea de almacén de la instancia en los nodos de la red. Al ser un inicializador, nos aseguramos que se va a ejecutar exactamente un sola vez y al inicio de la simulación.

- Estructura de datos: La estructura de datos es sencilla y la instancia básicamente consta de una matriz de adyacencia con los valores de los pesos del grafo correspondiente a la instancia y una variable que guarda el número total de ciudades de la instancia.
- Los métodos de la clase son estos,

  ```
  public int numCities()
  ```

  El método *numCities* devuelve el número total de ciudades que tiene la instancia, este método, como podemos suponer, será uno de los grandes utilizados por las demás clases de la biblioteca

  ```
  public double distance(int x,int y) throws ArrayIndexOutOfBoundsException
  ```

  El método *distance* devuelve la distancia que existe dos ciudades dadas.

  ```
  public double tourValue(int tour[ ]) throws ArrayIndexOutOfBoundsException
  ```

  Este método calcula la distancia de un camino dado en el parámetro de entrada, dicho camino tiene que ser un camino bien formado, es decir, tiene que tener la longitud de una camino completo para la instancia actual y su regreso al punto de partida, esto es, la longitud del camino se corresponde con el valor devuelto por el método *numCities*, en caso contrario lanza una excepción.

4.2.2.- LA CLASE FileETSP.

Esta clase crea instancias del problema de viajante de comercio que cumplan con la geometría euclidiana. Hay que añadir que la mayoría de casos de instancias, reales y de estudio, suelen ser TSP simétricos y euclidianos. Esta clase, como heredera de la anterior, comparte constructores y métodos que, en este caso, algunos son redefinidos para adaptarlos a esta geometría en particular donde hay que introducir el concepto de coordenadas, que situarán las ciudades en un espacio euclidiano, y calcular los valores de la matriz de adyacencia

respecto a la posición de las ciudades en coordenadas y obteniendo su distancia por la fórmula de distancia euclidiana, vista en el apartado 2.1.2 de la presente memoria.

Todos los métodos y constructores de la clase asumen que el espacio euclidiano utilizado para todas las instancias que se definan, es un espacio de bidimensional y dicho plano comenzará en las coordenadas iniciales (0,0) y terminará en las coordenadas de la ciudad más alejada de las coordenadas iniciales.

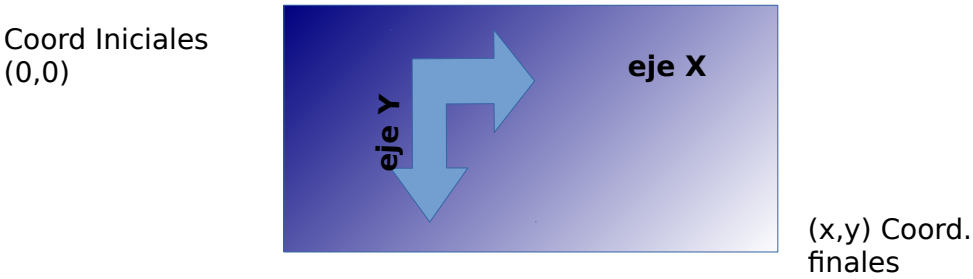


Fig. 4.01. Representación gráfica del espacio euclidiano válido para la clase.

Esta clase construye la instancia tras aceptar un fichero que tiene la forma estándar para instancias euclidianas bidimensionales del TSP que define el repositorio TSPLIB, un ejemplo de fichero es el siguiente:

```
NAME : d198
COMMENT : Drilling problem (Reinelt)
TYPE : TSP
DIMENSION: 198
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 0.00000e+00 0.00000e+00
2 5.51200e+02 9.96400e+02
3 6.27400e+02 9.96400e+02

N .........

196 4.02830e+03 1.08650e+03
197 4.02830e+03 1.01030e+03
198 3.95210e+03 1.01030e+03
EOF
```

La ruta y nombre del fichero se tiene que especificar en el fichero de configuración.

El siguiente método es nuevo y devuelve el valor de la coordenada indicada, 0 para el eje abscisa y 1 para el eje de ordenadas, para una determinada ciudad.

```
public double ithCoordenate(int i,int city) throws
ArrayIndexOutOfBoundsException
```

4.2.3.- LA CLASE RamdomETSP

Esta clase hereda de la clase DataTSP y crea una instancia con número aleatorios. Esta clase exige que en el fichero de configuración se defina el número máximo que tendrá una coordenada del espacio euclidiano definido. El cual coincidirá con el máximo de los número aleatorios que se generen.

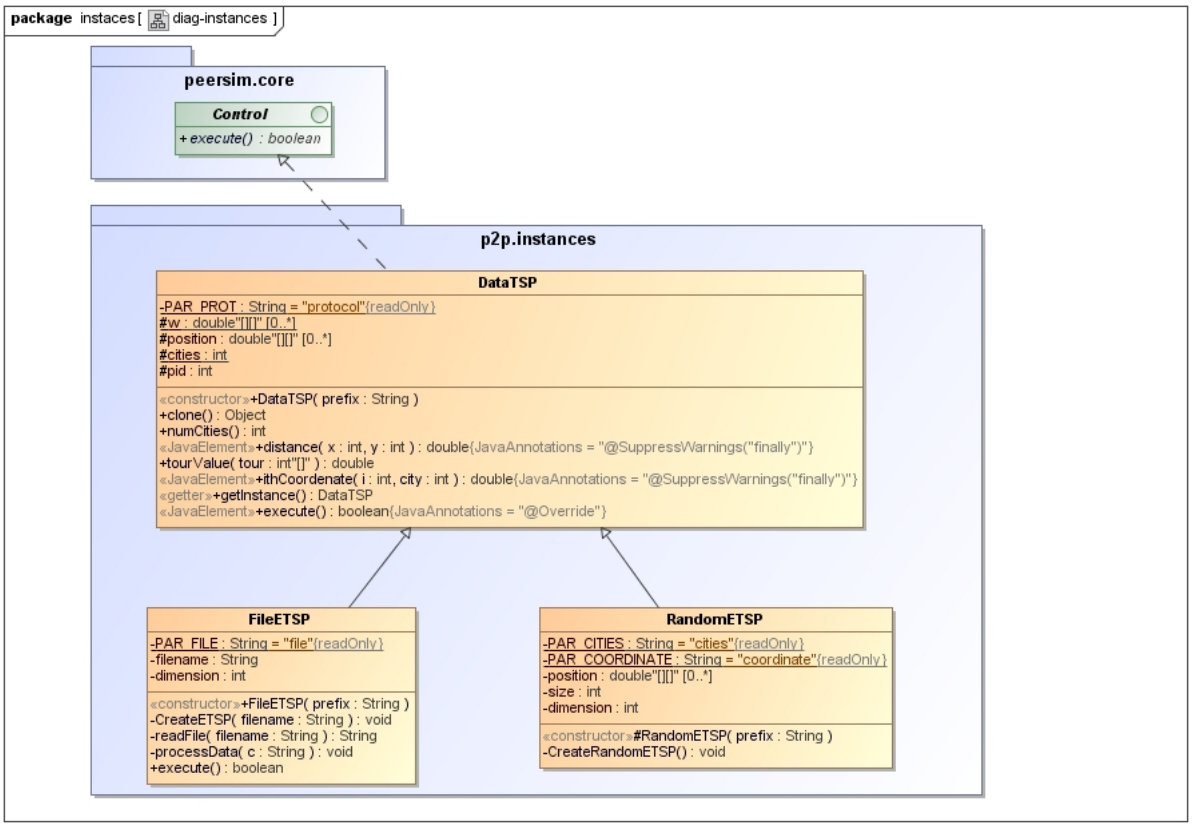


Fig. 4.02. Diagrama UML de clase para el paquete Instances.

4.3.- HEURÍSTICOS DE CONSTRUCCIÓN.

4.3.1.- CLASES PADRES.

La clase padre que engloba a los heurísticos de construcción implementados en el presente trabajo, es una clase abstracta. Una clase abstracta en JAVA es una clase donde, al menos, uno de sus métodos es abstracto o no tiene implementación, es decir, se define la cabecera del método pero no implementa ningún código, esto es dejado para ser implementado por las clases hijas. Por lo que una clase abstracta no se instancia sino que se utiliza como clase base para la herencia. Esto tiene la utilidad, y aquí es usada de esta manera, de garantizar una cabecera de los métodos común a todos las clases hijas.

Esta clase tiene el nombre de *TspConstructionHeuristic*.

- Estructura de datos.

  Las variables de instancias de la clase más importantes que debemos tener en cuenta son las siguientes:

```
protected DataTSP problem;        // Instace of a TSP problem

protected int tour[];          //  Tour gets from a TSP instance

protected double tourCost;     // value of a TSP tour
```

  La variable *problem* guarda la instancia del problema del viajante de comercio que es dada por el constructor de la clase que más adelante veremos. La variable *tour* contendrá el camino que generará el heurístico de construcción. Mientras que la variable *tourCost* reflejará el valor del camino que crea el heurístico de construcción.

- Constructor. Hay un único constructor que toma como parámetro de entrada una instancia del TSP, luego el constructor se encarga de inicializar las variables de la clase.

```
public TspConstructionHeuristic(DataTSP t)
```

- Métodos. El método más importante de esta clase es el método abstracto *generate*, al ser abstracto no tiene implementación y es usado por las clases que heredan de ésta para definir el método que genera un tour empleando el heurístico que implemente la clase hija. El camino que crea este método es puesto en la variable *tour*. Puede que las clases implementen más de un método *generate* con diversos parámetros de entrada, donde podremos modelar el comportamiento del heurístico. De todas formas, las clases hijas garantizan la implementación de este método sin parámetros.

```
public abstract void generate();
```

  El último método que vamos a mencionar de esta clase es *getTour* que da el camino generado por el método anterior en la variable pasada por referencia a *getTour*.

```
public void getTour(int t[])
```

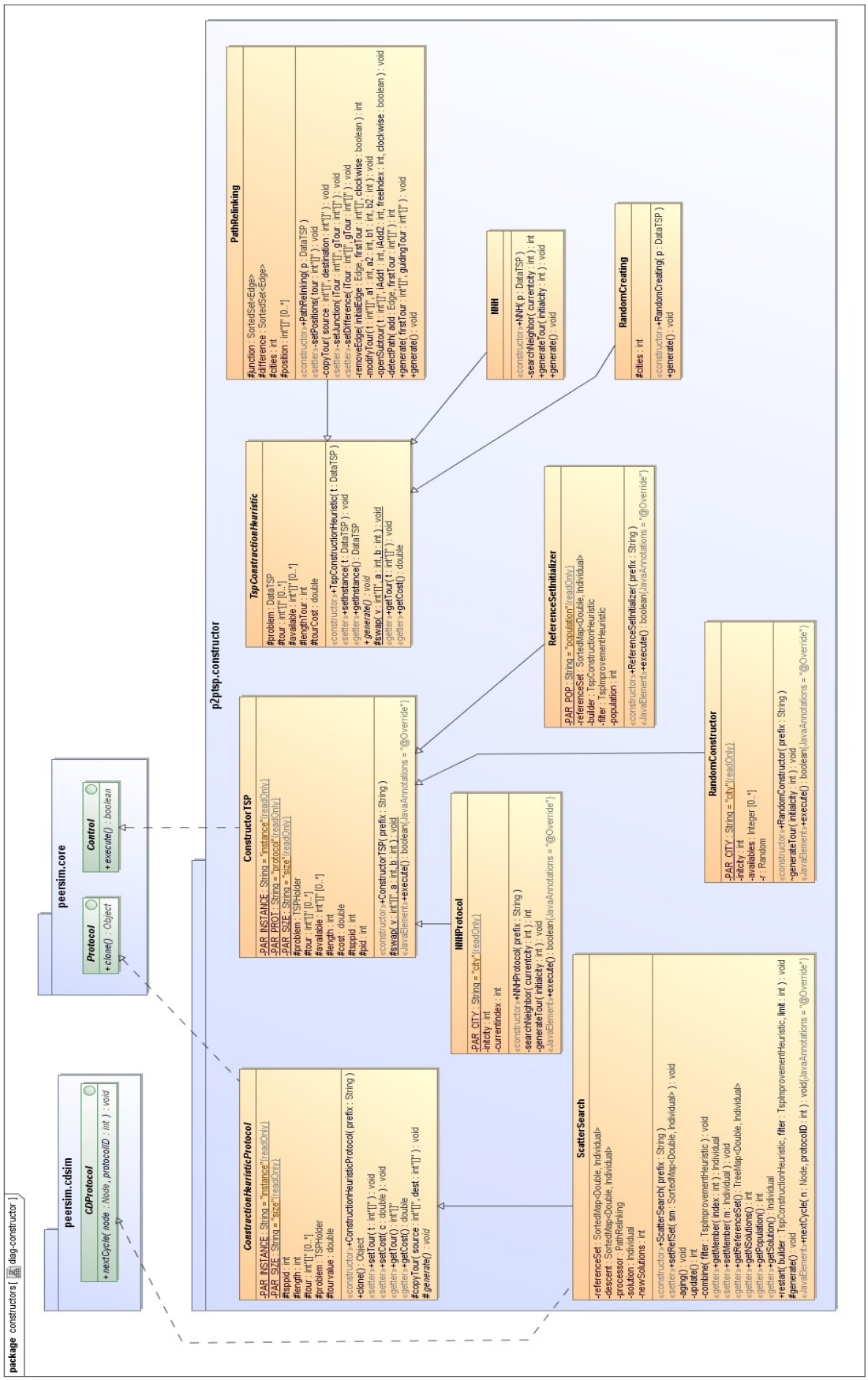


Fig. 4.03. Diagrama UML de clase para el paquete Constructor

Sin embargo, tal y como se puede ver en el diagrama anterior, esta no es la única clase padre que contiene el paquete *Constructor*. Se ha introducido de esta manera para facilitar la compresión inicial del diseño orientado a objetos que se ha empleado para codificar los heurísticos.

La anteriormente explicada y que sirve de referencia a las demás tiene su uso para los heurísticos que se han implementado sin una modificación para ser usados en un ambiente P2P. En otras palabras, dichas implementaciones se usarán como objetos dentro de otras clases que necesiten hacer uso de un heurístico de construcción.

La siguiente clase padre se diseña para cuando el heurístico de construcción tiene que cumplir su función como protocolo dentro de los nodos de la red P2P. Tiene la misma arquitectura que el anterior, ofreciendo los mismos métodos; tan sólo, se trata de una adaptación a PeerSim con la implementación de las interfaz *Protocol* y del método *clone()* que exige dicha interfaz. Puesto que, a modo de prototipo, tan sólo el primer nodo crea sus protocolos instanciado las clases de los mismos y el resto son copias de este prototipo.

La última clase padre se concibe cuando un heurístico de construcción ha de emplearse como un objeto inicializador de PeerSim. Este caso se da cuando los nodos han de ejecutar un heurístico de mejora, entonces el heurístico de construcción actúa como un inicializador donde ofrece al heurístico de mejora un tour inicial donde poder realizar su trabajo. Esta clase tiene que implementar la interfaz *Control* y escribir el método *execute().* Además esta clase es padre de una clase que es no es un heurístico, sino es una parte de uno llamado Scatter Search. Esta clase hija tiene como misión de inizializar el conjunto de referencia que usa el heurístico antes mencionado.

A continuación daremos más detalles del funcionamiento de este heurístico y los demás.

4.3.2.- CONSTRUCCIÓN ALEATORIA DE UN CAMINO.

El primer heurístico de construcción que veremos es muy sencillo e ingenuo en su planteamiento. Trata de construir un camino añadiendo las ciudades de la instancia del TSP dada, escogiéndolas de forma aleatoria e insertándolas en el camino que se va construyendo. El paquete *peersim.core.CommonState* es el encargado de generar los número aleatorios que designará a las ciudades candidatas a formar parte del tour. La única precaución que se debe tomar en

este heurístico, es no introducir dos veces la misma ciudad; pues esta situación puede darse al tratarse de números aleatorios, incumpliéndose el enunciado del problema del viajante de comercio, por tanto.

Como hemos estado comentando anteriormente, las clases hijas de la clase de heurísticos de construcción sólo tienen que implementar el código del método abstracto *generate*, más otros métodos internos, o privados en la terminología JAVA, que se requieran. En este caso, dada su sencillez no necesita ningún método más y a modo de ejemplo expondremos aquí, esta vez, el código fuente.

```
public void generate() {
        int i,value;
        boolean found;
        boolean randVector[];

    randVector=new boolean[cities];
    value=0;
    for (i=0;i<cities;i++)
        randVector[i]=false;
    for (i=0;i<cities;i++) {
        found=false;
        while (!found) {
            value=CommonState.r.nextInt(cities);;
            if (!randVector[value]) {
                randVector[value]=true;
                found=true;}}
        tour[i]=value;}
```

Como este método es muy sencillo y su codificación es trivial y no presenta problemas, lo usaremos para ver la diferencia entre la implementación que se ha hecho como heurístico normal y su adaptación a la red P2P. En este caso como inicializador.

```
private static final String PAR_CITY="city";

private int initcity;

initcity = Configuration.getInt(prefix + "." +
PAR_CITY,CommonState.r.nextInt(length));
```

El constructor del inicializador como se ejecuta al principio de la simulación y desconoce en ese momento si el planificador de PeerSim ha construido la instancia del TSP o si está disponible para consultarla. Requiere que en el fichero de configuración de la simulación se le informe del número de ciudades que

compone la instancia, donde se infiere la longitud del tour a generar. En primer lugar, se define una constante que le dará nombre al parámetro que se tiene que poner en el fichero de configuración, en este caso, es PAR_CITY. Luego se declara una variable que contendrá el valor que se especifique en el fichero de configuración.

Se codifica mediante la API de Java de Properties, la lectura desde el fichero de configuración de dicho parámetro. En caso, de que no viniera definido el parámetro se toma un valor al azar.

A continuación se escribe el método *generate()* que implementará la acción del heurístico.

Finalmente, teniendo en cuenta que es un inicializador, se deberá implementar el método *execute()*, llevando a cabo en este caso, la adaptación del heurístico al entorno P2P.

```
@Override
public boolean execute(){

               ImprovementHeuristic ih;
               int i;
               double v;

            problem=(TSPHolder)
Network.get(0).getProtocol(tsppid);
            generateTour(initcity);
            v=problem.tourValue(tour);
            for (i = 0; i < Network.size(); i++) {
                  ih = (ImprovementHeuristic)
Network.get(i).getProtocol(pid);
                  ih.setTour(tour);
                  ih.setCost(v);
            }
            return false;
       }
```

Cuando el planificador llama al método *execute()*, sí existe garantía del que el protocolo al que inicializa se encuentra totalmente construido. Esto quiere decir que ya está disponible la instancia del TSP para el problema a resolver y también está disponible el algoritmo de mejora al que este inicializador le va a servir un tour inicial.

Las primeras líneas declaran una variable – *ih* – para contener la instancia del heurístico de mejora y en – *problem* – se comunica con un nodo de la red para que le devuelva su instancia del TSP. Ahora está en condiciones de generar un tour para dicha instancia.

Una vez creado el tour, debe comunicar con todos los nodos de la red y transmitírselo al protocolo que implemente el heurístico de mejora de dichos nodos. Además les informa del valor del camino generado.

Por último, observamos que el método *execute()* debe devolver una variable booleana que en este caso su valor es falso. Esto se debe a que los inicializadores son objetos de tipo Control de PeerSim y tienen la facultad de detener la simulación bajo las circunstancias que se consideren. El planificador recibe la orden de parada por parte de estos objetos Control desde el valor devuelto por dicho método *execute()*. Como estamos ante el caso de un inicializador no tiene sentido que detenga la simulación, por este motivo, siempre devuelve falso.

4.3.3.- ALGORITMO VORAZ: EL VECINO MÁS CERCANO.

Este tipo de algoritmos son llamados voraces, ávidos o su nombre en inglés *greedy*. Son heurísticos muy sencillos en su planteamiento e implementación, que pueden ser aplicados a numerosos problemas, sobretodo a los de optimización.

En general, se describen para que un problema pueda ser resuelto mediante un método voraz tiene que identificarse en el una serie de elementos; de este método general expondremos su aplicación concreta para el problema del viajante de comercio:

- Un conjunto de *candidatos*. En nuestro problema son las ciudades que todavía no forman parte de la subsolución que va creando el algoritmo. Al inicio serán todas las ciudades de la instancia.
- Una función que compruebe si un cierto subconjunto de candidatos es *factible*. El algoritmo, en el TSP, va incorporando las ciudades candidatas de una en una; se determina si es factible de ser incorporada a la solución propuesta si dicha solución no impide que sea un “camino bien formado”, esto es, no introduzca ciclos sin estar todas las ciudades presentes en la solución o que la ciudad a incorporar eleve el número de aristas por ciudad a un número mayor que 2, ya que, en un camino correcto cada ciudad tiene exactamente 2 aristas.
- Una función *objetivo* que determine el valor de la solución hallada. Es la función que queremos maximizar o minimizar. En particular para el TSP, la función objetivo es la suma de los pesos de las aristas del camino construido, como sabemos queremos minimizar dicha suma.

- Una función que compruebe si un subconjunto de estas entradas es solución al problema, se óptima o no. Para el TSP, este apartado es bastante difícil de implementar por la misma naturaleza del problema, recordemos que es un problema NP-arduo y conocer la optimalidad es intratable. Hay que ceñirse a realizar una demostración formal -normalmente por reducción al absurdo- o fijarse en el repositorio TSPLIB y estamos con instancias que pertenecen a dicho repositiorio. Por lo que, el algoritmo voraz no implementa esta parte.

Una vez visto que se cumplen los requisitos para emplear esta técnica, damos a conocer la estrategia que seguirá el algoritmo voraz: dicho algoritmo incorporará la ciudad de partida del problema a la solución que va a empezar a formar, una vez hecho esto escogerá entre todas las aristas o caminos que salen (o conectan dicha ciudad de partida con otras) la tenga un peso menor de todas ellas, esto es, el conjunto de candidatos, incorpora dicha arista y ciudad que conecta a la subsolución y repetirá todo el proceso anterior considerando las restricciones del problema mediante la función factible, por último, una vez incorporada todas la ciudades, se añade el camino de vuelta a la ciudad de partida.

Como vemos, este heurístico tiene que ser muy veloz en su tiempo de ejecución y eficiente en el uso de los recursos del sistema. Ciertamente es así, porque su complejidad computacional pertenece a $O(n^2)$ y su estrategia parece muy convincente y en una primera aproximación al problema seguramente nosotros mismo idearíamos una solución parecida.

Sin embargo, en el siguiente ejemplo veremos que este método no siempre funciona pues cuando toma una decisión, ya queda incorporada a la solución para siempre y no mira más allá que en el punto exacto de donde se encuentra, en ese momento, del grafo teniendo entonces una visión local del problema.

Este método también es conocido por su nombre en inglés y que le da nombre al código implementado en la biblioteca, *Nearest Neighbour Heuristic (NNH)*. El código tiene un método que crea un camino dando una ciudad de partida como parámetro de entrada y luego el método de la clase padre *generate* que toma una ciudad al azar para empezar.

```
public void generate(int initialcity)
```

	a	b	c	d	e	f
a	∞	3	10	11	7	25
b		∞	6	12	8	26
c			∞	9	4	20
d				∞	5	15
e					∞	18
g						∞

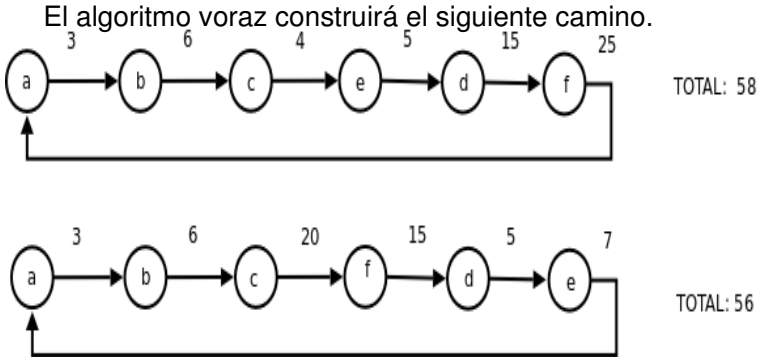


Fig. 4.04. Ejemplo de ejecución del heurístico del vecino más cercano.

4.3.4.- HEURÍSTICO PATH-RELINKING.

En español podríamos llamar a esta técnica como heurístico de re-encaminamientos de trayectorias o re-enlazado de caminos. Sin embargo, puede ser por la novedad de este heurístico, es más conocido por su nombre en inglés y sus siglas también, *Path Relinking (PR)*, donde en este documento haremos referencia a este método por su nombre en inglés.

Path Relinking fue propuesto originalmente en el contexto de la búsqueda tabú por Fred Glover en los años 70. Sin embargo, el desarrollo e investigación sobre esta heurística se encuentra en la actualidad en pleno apogeo, sobretodo en la segunda mitad de la década de los 90 y esta primera década del siglo XXI; motivada quizás por la mejora en la capacidad de cómputo de los procesadores y nuevas arquitecturas y distribución de computadores que propician encarar el estudio de estas materias. Junto a Glover otros muchos autores, de distintos países, continúan publicando diversos artículos científicos sobre este heurístico propiamente o como parte en otros métodos.

En esta memoria se ha incluido a Path-Relinking en el grupo de los heurísticos de construcción porque, dicho de forma coloquial, en algún sitio había que ponerlo. Esto no es así, exactamente, pero tampoco se trata de un heurístico de mejora en términos exactos y se encuentra bastante lejos de serlo. No obstante, los usos a los que se somete Path-Relinking -sobretodo, su aplicación junto a otros heurísticos- nos hacen más identificarlo con un heurístico de construcción. Esta dificultad en clasificarlo viene porque Path-Relinking para el problema del viajante de comercio no tiene una aplicación directa por el mismo. Esto es, dada una instancia del TSP no construirá un

camino y dado un único camino no encontrará un camino que mejore éste. Este heurístico toma como entrada a dos caminos y da como resultado a un camino que es igual o mejor que los dados; por sí sólo no tiene una gran aplicación pues sus resultados, de esta forma, no tienen mucho interés para el TSP.

Por este motivo, Path-Relinking es utilizado, normalmente y para el problema del viajante de comercio, dentro de otros heurísticos donde puede ser mejor aprovechado su potencial. De hecho, la razón de implementar este heurístico para esta biblioteca es para poder desarrollar la siguiente técnica que se presentará, Búsqueda Dispersa (*Scatter Search)*. Incluso en la literatura que se puede encontrar para el TSP suele ser un binomio Path-Relinking y Búsqueda Dispersa.

Ahora veremos con un poco de detalle cómo funciona este método. Como hemos indicado anteriormente el algoritmo toma dos caminos bien formados para una determinada instancia del TSP. A uno de ellos lo llamaremos, según la terminología para el Path Relinking, **camino inicial** o, en inglés, *initial tour*. Al otro camino se le llamará **camino guía** o, en inglés, *guiding tour*. Este camino inicial sufrirá una serie de modificaciones sucesivas para dar al final como resultado al camino guía que se ha dado como entrada. Esto es, el tour inicial se transformará al final del Path Relinking en el camino guía. Por tanto, el camino inicial es objeto de las modificaciones y es el principal actor en el algoritmo; mientras que el tour guía tiene un papel secundario y sirve sólo como objetivo o reflejo a lo que tiene que convertirse el tour inicial.

Todas las transformaciones que ocurran con el camino inicial en cada iteración del algoritmo tendrá siempre que dar a un camino bien formado o válido para la instancia del TSP en la que estamos trabajando. De esta forma, en cada iteración vamos obteniendo nuevos caminos a los que se le calculará su valor y se compararán con el valor del camino inicial, camino guía y todos aquellos que se hayan generado en cada iteración.

La salida del Path Relinking será el camino, de todos lo generados, cuyo valor sea mínimo, tal y como, especifica el problema del viajante de comercio.

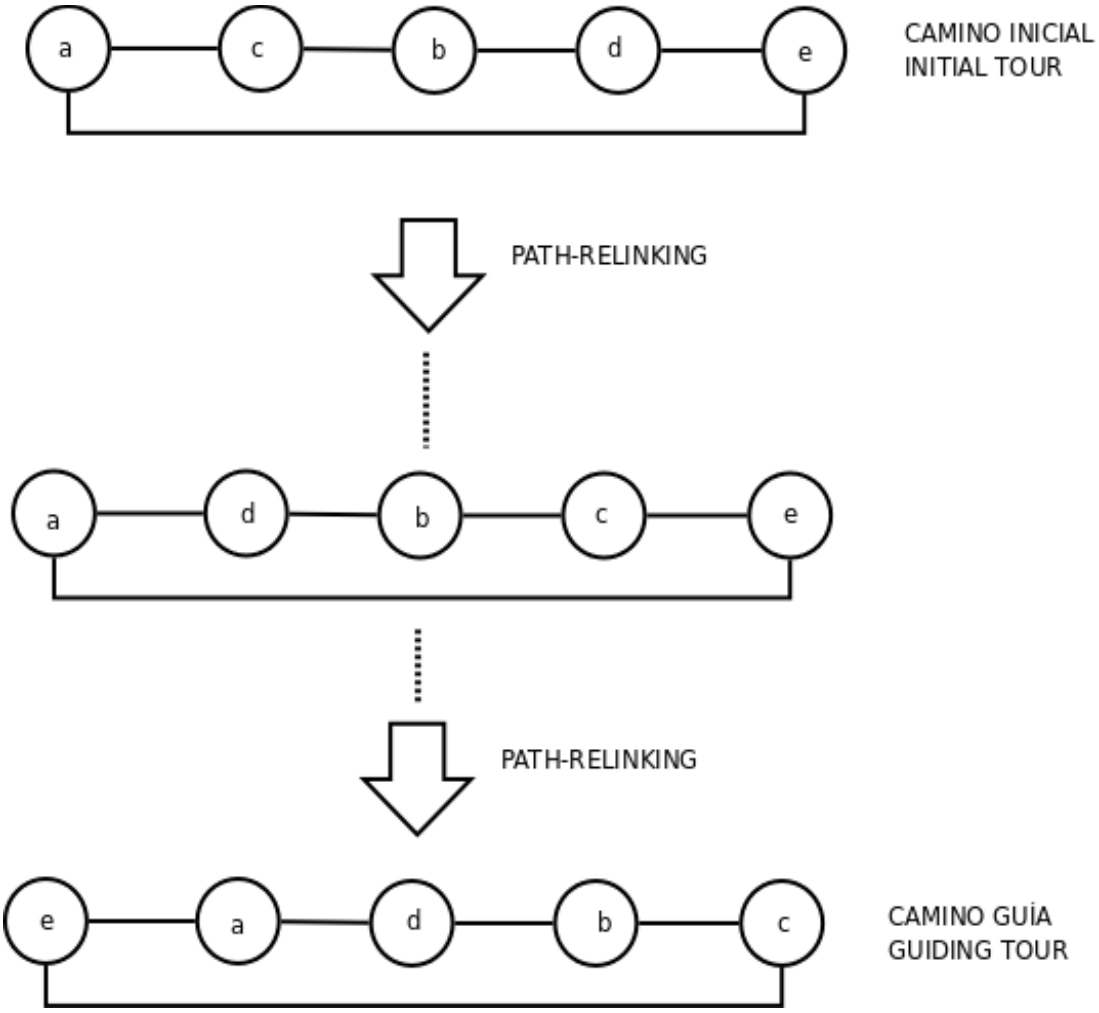


Fig. 4.05. Traza de la evolución del algoritmo de Path Relinking

En este punto tenemos ya una clara idea de la filosofía del heurístico. Para comprender su completo funcionamiento nos queda por definir cuales son esas operaciones o procedimientos que componen esas “transformaciones” que sufre el camino inicial.

A modo de paréntesis, previamente, vamos a tener en cuenta a dos conjuntos. El conjunto A que estará formado por las aristas del camino inicial. Y el conjunto B que contendrá a las aristas del camino guía. Estos dos conjuntos no tienen una implementación en el algoritmo y solamente cumplen una función matemática para definir a los dos siguientes conjuntos que sí tienen relevancia para esta técnica.

En primer lugar tenemos que mencionar dos conjuntos de aristas que el Path Relinking necesita para su funcionamiento y que son muy importantes para comprenderlo. Un primer conjunto, es el **conjunto de las uniones** *(junction set),* que se obtiene de realizar la operación unión a los conjuntos A y B, (*juntion*

= A U B), esto es, contendrá las aristas que son comunes entre el camino inicial y el camino guía en el inicio del algoritmo. El segundo conjunto, **conjunto diferencia** *(difference set)*, nace de aplicar la operación de conjuntos diferencia, *(difference = B – A)*, que albergará las aristas que están en el camino guía y no están en el camino inicial; esto es, las aristas no comunes del camino guía con el camino inicial al inicio de la ejecución del algoritmo. Gráficamente podemos verlos en la siguiente ilustración para los caminos de la figura 4.06.

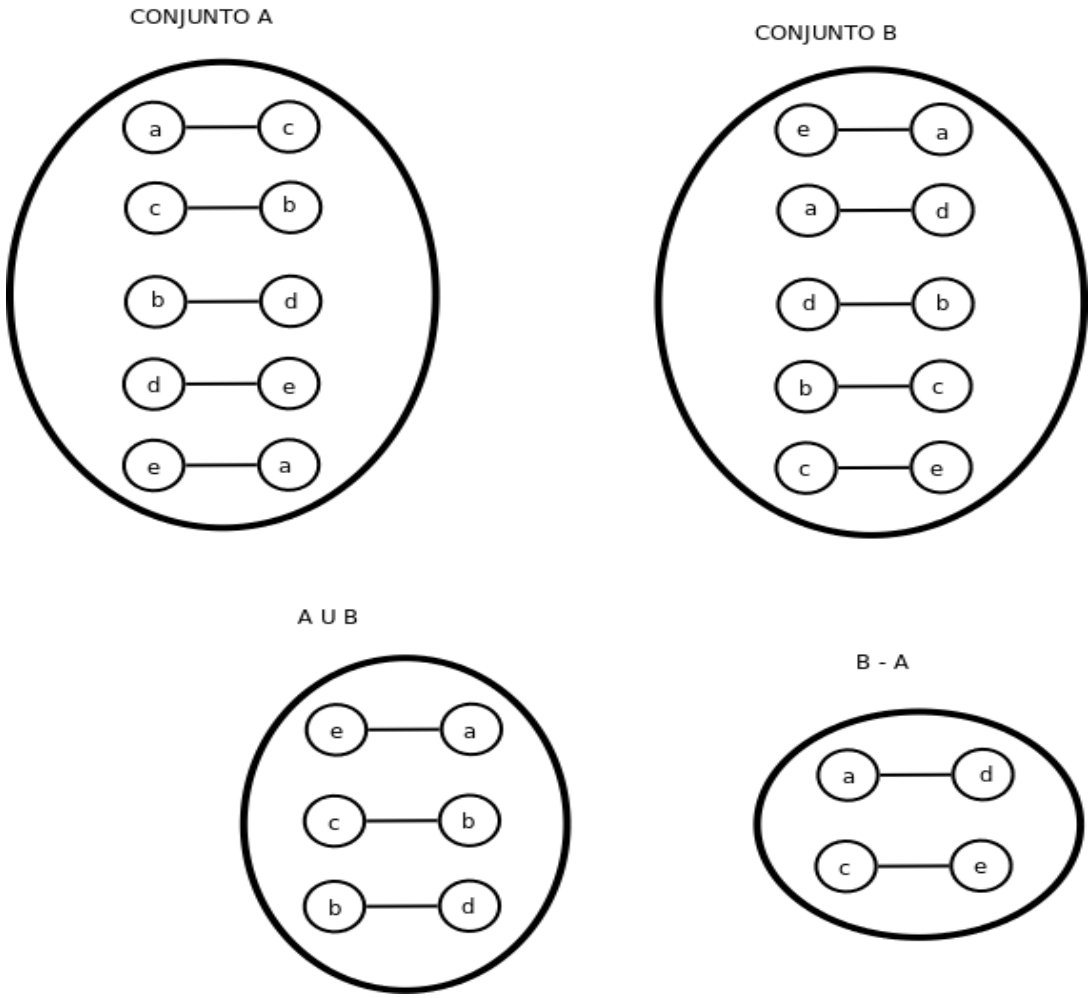


Fig. 4.06. Operaciones sobre conjuntos

El conjunto diferencia tiene como misión ofrecer en cada iteración del algoritmo una arista, la cual, recordemos, contiene el camino guía y no está en el camino inicial, que será insertada en el camino inicial que es el que sufrirá la modificaciones hasta llegar a ser igual que el camino guía. De esta forma, iremos incluyendo en el camino las aristas que les falta para ser igual que el camino guía. La arista agregada al camino también es incluida en el conjunto unión, pues ya es una arista común a los dos caminos, inicial y guía. Consecuentemente, esta arista es borrada del conjunto diferencia pues ya no es una arista que esté presente en

el camino guía y no lo esté en el camino inicial.

Así pues, en cada iteración el cardinal del conjunto unión aumenta en uno y el cardinal del conjunto diferencia disminuye, también, en una unidad.

El conjunto unión tiene como cometido el servir como referencia para saber las aristas que hay que borrar del camino inicial cada vez que se introduce una arista procedente del conjunto diferencia. Tenemos que tener en cuenta que el camino inicial es un camino válido o bien formado para la correspondiente instancia del TSP que se esté utilizando. Por tanto, si en cada iteración se le añade una arista más, entonces el camino deja de ser un camino válido; pues recordemos que para una instancia de n ciudades, un camino hamiltoniano para dicha instancia tiene que tener exactamente n aristas. Por este motivo, cada vez que una arista es incorporada al camino que se está modificando, una arista debe ser eliminada.

El criterio para eliminar una arista del camino es el siguiente. En el camino inicial o el camino en que el que se haya convertido tras las sucesivas iteraciones del algoritmo existen dos grupos de aristas: las aristas que están en el conjunto unión y, por tanto, son comunes a los dos caminos, inicial y guía, o las otras aristas que no son comunes y no están en el camino guía. Estas últimas son las candidatas a ser eliminadas del camino pues con ellas nunca llegaremos hasta el camino guía. Así, en cada iteración y, por ende, cada vez que se añade una arista del conjunto diferencia, una arista del camino que se transforma que no es común al camino guía es eliminada. Para saber que una arista es candidata a ser eliminada, se comprueba que dicha arista no pertenezca al conjunto unión, puesto que, de otra forma inexcusablemente tiene que ser una arista no común al camino guía.

De esta forma, se mantiene el número de aristas correcto para que el camino resultante sea un camino hamiltoniano en cada iteración.

En consecuencia, si en cada iteración una arista que está en el camino guía y no en el camino inicial es incluida en éste último y, en esa misma iteración, es eliminada una arista que no es común a los dos caminos y no está en el camino guía, entonces habrá un momento durante la ejecución del algoritmo en el que camino inicial llegue a ser exactamente igual que el camino guía.

Este momento es el que define que criterio de parada del método que es cuando el camino inicial se convierta en el camino guía y que justo coincidirá cuando el conjunto diferencia sea un conjunto vacío, ya que, como hemos

mencionado anteriormente el número de elementos del conjunto diferencia decrece durante la ejecución del algoritmo.

Para resumir todas estas ideas dejamos a continuación en pseudocódigo el procedimiento inicial de esta técnica:

```
TOUR PATH_RELINKING (INICIAL:TOUR, GUÍA:TOUR)

VARIABLES
     UNION,DIFERENCIA TIPO CONJUNTO;
     MEJORCAMINO TIPO TOUR;
     MEJORVALOR, VALORCAMINO TIPO FLOTANTE;
     A,E TIPO ARISTA;

INICIO
     UNION=CREAR_UNION(INICIAL,GUÍA);
     DIFERENCIA=CREAR_DIFERENCIA(INICIAL,GUÍA);
     MEJORCAMINO=INICIAL;
     MEJORVALOR=CALCULAR(INICIAL);

     MIENTRAS ( DIFERENCIA ≠ Ø ) HACER
          A = DIFERENCIA.DARELEMENTO();
          DIFERENCIA.BORRAR(A);
          UNION.AÑADIR(A);
          INICIAL.AÑADIR(A);
          E=ESCOGER UNA ARISTA DE INICIAL NOTє UNIÓN
          INICIAL.ELIMINAR(E);
          SI INICIAL.VALOR() < MEJORVALOR ENTONCES
               MEJORCAMINO=INICIAL;
               MEJORVALOR=INICIAL.VALOR();
          FIN_SI
     FIN_MIENTRAS

     DEVOLVER MEJORCAMINO;
FIN PATH_RELINKING
```

Entraremos a continuación en algunos detalles de la implementación que se ha llevado a cabo para la construcción de la biblioteca de heurísticos para este trabajo y que merece la pena conocer, pues nos parece importante y nos aclarará el funcionamiento de este método.

Como vemos, la operación de eliminar la arista se ha dejado después de insertar la arista correspondiente. Esto es así porque de esta manera es más fácil decidir qué arista será la candidata a ser borrada, ya que, cuando se inserta la arista se produce un lazo que hay que romper. Esta situación podemos representarla de la siguiente manera en la figura 4.07. introduciendo la arista (a,b) en el camino inicial representado por un círculo.

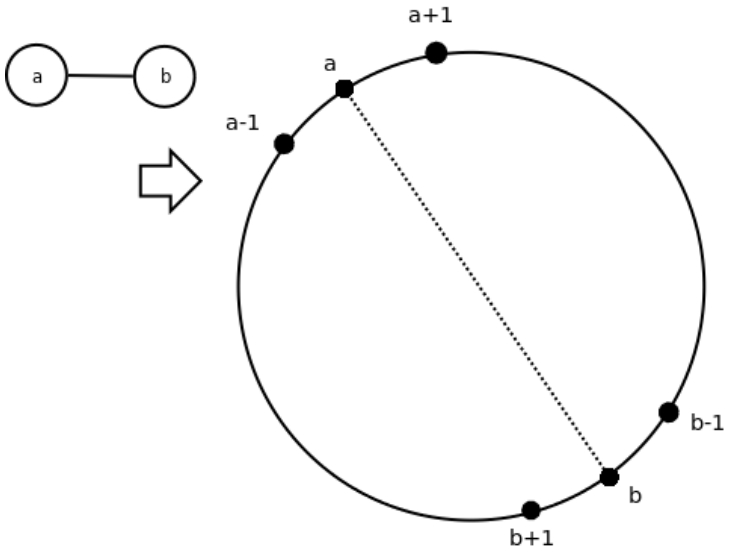


Fig. 4.07. Inserción de la arista (a,b) por Path Relinking.

Al introducir la arista (a,b) tenemos que tener en cuenta que aparecen nuevas, y estas son (a-1, a), (a, a+1), (b-1,b) y (b, b+1). Siendo “a-1” la ciudad que se encuentra una posición anterior a la posición de la ciudad “a” en el camino. De forma parecida ocurre con “a+1” que es la ciudad que se encuentra una posición posterior a la posición de la ciudad “a” en el camino. Y, análogamente, ocurre con las nomenclaturas “b-1” y “b+1” para la ciudad “b” del camino.

Estas nuevas cuatro aristas serán entonces las candidatas a eliminar; de hecho, serán dos de ellas las necesarias a eliminar para que el camino continúe cumpliendo la condición de ser un camino hamiltoniano. Antes de eliminar a cualesquiera de estas cuatro aristas hay que comprobar que dichas realmente son prescindibles, esto es, no sean aristas que pertenezcan al conjunto unión y que, por tanto, impedirían el objetivo de Path Relinking de alcanzar el camino guía si se eliminaran, estas comprobaciones serán realizadas por el procedimiento especificado en el pseudocódigo como “escoger una arista que no pertenezca al conjunto unión”.

Teniendo en cuenta todas estas consideraciones se nos presentan cuatro situaciones, dos de ellas fáciles de resolver -eliminando los pares (a-1,a) y (b-1,b) ó (a,a+1) y (b,b+1)- y otras dos más complicadas de llevar a cabo y que el algoritmo ha de resolver -eliminando los pares (a-1,a) y (b,b+1) ó (a,a+1) y (b-1,b). Veámoslo en la siguiente ilustración.

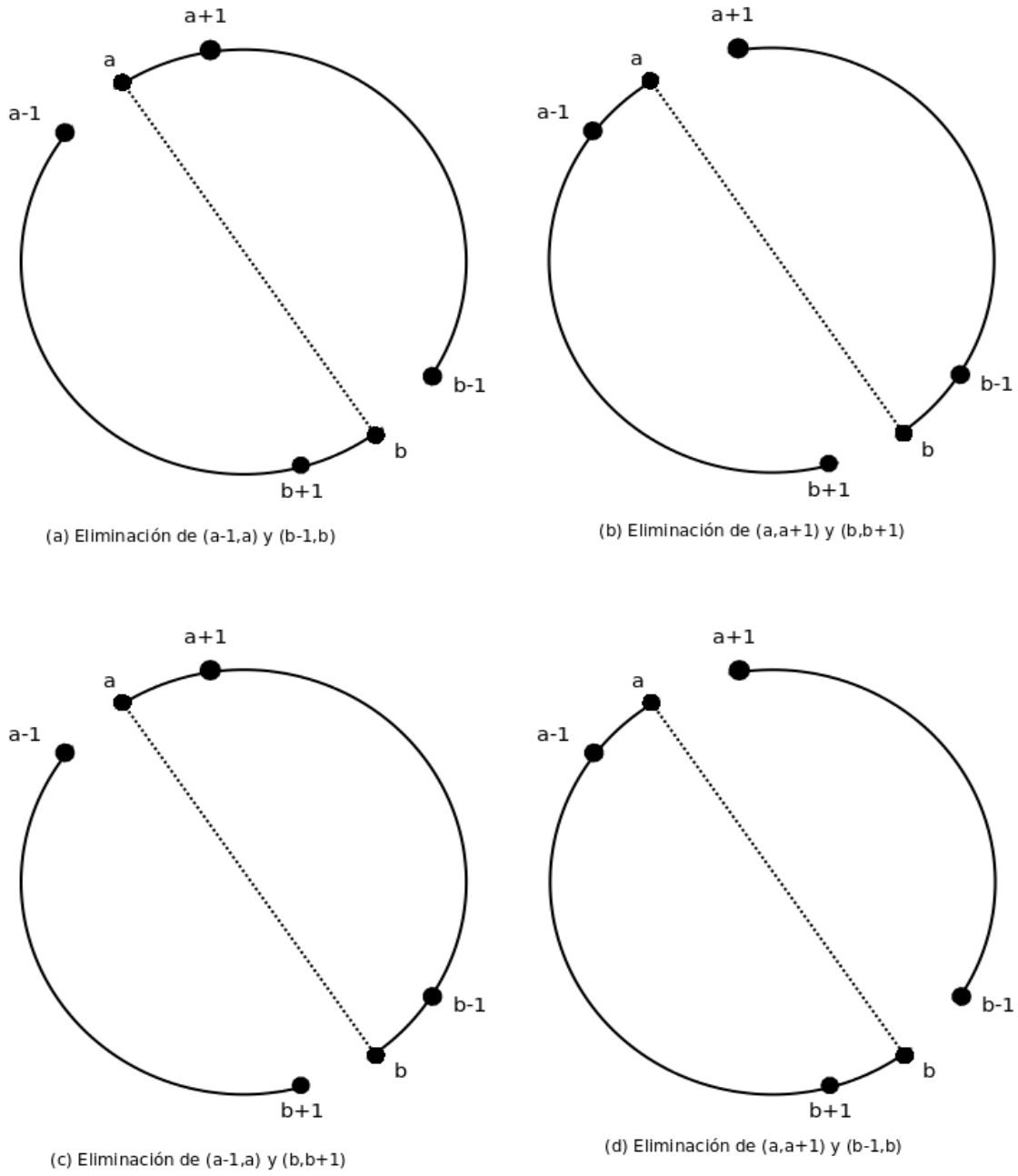


Fig. 4.08. Posibles situaciones del camino al eliminar aristas en Path Relinking.

Siendo un poco observadores nos percatamos que hemos añadido una sola arista pero hemos eliminado dos del camino. Efectivamente, habrá que añadir una arista más que dará la oportunidad de volver a convertir el camino en transformación en un camino hamiltoniano. Especialmente, esto es así en los casos (a) y (b) de la figura 3.11 los cuales se corresponde con las situaciones más fáciles de resolver que mencionábamos en líneas anteriores. Puesto que, en la situación (a) se agregará la arista (a-1,b-1) y en la situación (b) se añadirá la arista (a+1,b+1), esto es, las ciudades o vértices de grado 1 o a las que sólo les llega una sola arista.

De esta manera, se da por finalizada la transformación del camino inicial en la iteración actual y se continúa con la siguiente comprobando antes el criterio de parada del algoritmo.

Las situaciones (c) y (d) de la figura 3.11 no pueden resolverse de la forma anterior, pues nos encontramos con un lazo interior que divide el camino inicial en un subcamino cíclico y otro subcamino separado de éste último y que no contiene ciclos, tal y como podemos ver en la situación (d) de la figura 3.11. Si unimos las ciudades que tienen grado 1 -por ejemplo, en la situación (d), agregando la arista (a+1,b-1)- obtendremos dos subcaminos cíclicos, llegando entonces a un estado anómalo para el Path Relinking, el cual no tiene solución.

Por lo que habrá que imaginar una solución que siga la técnica de Path Relinking, de obtener en cada iteración un camino hamiltoniano. Recordemos en estos casos (c) y (d) hemos añadido una arista y eliminado dos, así que, la solución que propongamos tendrá que equilibrar este balance de aristas. En la biblioteca implementada para este proyecto se ha optado por volver a eliminar una arista más del subcamino cíclico que ha surgido.

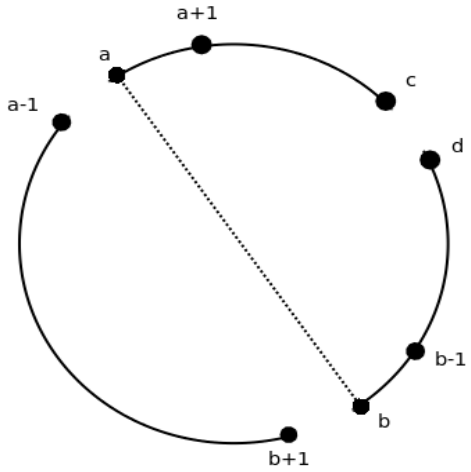


Fig 4.09. Eliminación de una nueva arista en Path Relinking.

Como vemos en la figura 4.09, que recrea la situación (c) de la figura 4.08, se ha encontrado en el subcamino cíclico una nueva arista que es eliminada, esta arista es la (c,d). Por supuesto, esta nueva arista tampoco puede pertenecer al conjunto unión. Ahora mismo, el número de aristas eliminadas se eleva a tres y continuamos con una sola arista añadida.

El desequilibrio en este balance de aristas añadidas y borradas será subsanado con los siguientes pasos. Ya que, solamente nos queda agregar dos aristas, según la figura 4.09, (a-1, c) y (b+1,d) o, viceversa, (a-1,d) y (b+1,c);

atendiendo a la idoneidad de las arista para formar parte del camino. Así pues, queda saldada la cuenta con estas dos aristas incorporadas, con tres aristas eliminadas y tres aristas añadidas.

Como hemos dicho al inicio de este apartado, en este trabajo usamos Paht Relinking para implementar otro heurístico, Búsqueda Dispersa y que en el apartado siguiente vamos a exponer esta combinación técnicas que es ampliamente usada y que nacieron casi de la mano.

En la biblioteca implementada el método “generate” de Path Relinking tiene la siguiente cabecera:

```
public void generate(int firstTour[ ],int guidingTour[ ])
```

Como siempre también está el método común a todos los heurísticos de construcción sin argumentos de entrada.

4.3.5.- BÚSQUEDA DISPERSA (SCATTER SEARCH).

Búsqueda dispersa, como es traducido en español, es el último de los heurísticos de construcción que se han implementado para la biblioteca desarrollada para este proyecto fin de carrera. Es muy conocido también por su nombre en inglés, *Scatter Search (SS)*. Usaremos las dos denominaciones indistintamente.

Esta técnica aparece descrita por primera vez en 1977 por Fred Glover. Al igual que le ocurrió a Path-Relinking, recordemos del mismo autor, no ha sido a partir de los años 90 del pasado siglo hasta la actualidad cuando ha experimentado un aumento en el interés de la comunidad científica por este método.

Su propio autor publicó en 1998 una modificación del heurístico, la cual se considera actualmente como la versión estándar o de referencia, el método original que describió en los 70 y los autores que se dedican a su estudio están proponiendo nuevas aproximaciones del heurístico. Por tanto, debemos considerar en la actualidad a Scatter Search como un método en plena fase de desarrollo. Son muchos de los aspectos de este heurísticos que deben determinarse y depurarse, como el tamaño óptimo de las poblaciones, técnicas o heurísticos más apropiados para cada una de sus fases, regeneración de nuevas poblaciones, … que veremos según detallemos esta técnica.

Búsqueda dispersa es un método evolutivo o poblacional y aunque es

menos conocido que los algoritmos genéticos; se está aplicando con éxito a numerosos problemas difíciles de optimización. Para el problema de viajante de comercio, sobretodo en el capítulo cuarto de la presente memoria, podremos conocer los sorprendentes resultados en la aplicación de este heurístico.

Como Scatter Search va a ser el único representante en la biblioteca objetivo de este trabajo fin de grado de este tipo de técnicas basadas en los postulados de la evolución biológica y, más concretamente, en los algoritmos evolutivos. Vamos brevemente a presentar una pequeña introducción a estos métodos.

Las técnicas heurísticas computacionales basadas en el paradigma de la evolución biológica pertenecen y son una rama de la inteligencia de artificial y cada una de ellas tratan de resolver problemas haciendo una simulación artificial de algún proceso natural de la evolución de los seres vivos.

En concreto, los algoritmos evolutivos están diseñados para resolver problemas de optimización y/o búsqueda de soluciones en espacios extensos y no lineales, en donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable.

Esta técnica trata de resolver problemas imitando el sistema de evolución de las poblaciones de seres vivos. Existe un conjunto de entidades que representan posibles soluciones al problema, las cuales se mezclan y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer en el tiempo, evolucionando a mejores soluciones cada vez.

Existe una terminología para estos algoritmos que se corresponde con la empleada en los procesos evolutivos biológicos. Las entidades que representan a las posibles soluciones son llamadas **individuos**. Al conjunto de individuos se le denomina **población**. Los individuos son modificados por operadores genéticos, como el **cruzamiento** -mezcla de información de dos o más individuos, la **mutación** -es un cambio aleatorio en los individuos- y la **selección**, que consiste en la elección de los mejores individuos o aptos y que sobrevivirán a la siguiente **generación**. Con la selección se asegura que continúen los individuos o soluciones más apropiadas para el problema y la población a mejorando gradualmente.

Scatter Search es un método que sigue bien este paradigma de los algoritmos evolutivos expuesto. La diferencia fundamental es que Búsqueda dispersa no contempla originalmente la operación de mutación sobre los

individuos, es una técnica mucho más sistemática.

De igual manera, la operación de cruzamiento no existe como tal. En su lugar existe una operación de **combinación** a la que podemos tratar como equivalente al cruzamiento pero su comportamiento no es exactamente igual. Durante la corta historia de Scatter Search la operación de combinación ha cambiado, pues admite distintas técnicas que se están probando en este método aún en desarrollo, como la combinación ponderada, combinación basada en votos, etc. Algunos autores han propuesto recientemente como operador de combinación a Path Relinking y es el que se ha implementado en esta biblioteca. Una de las ventajas del uso de Path Relinking es la independencia de la combinación de soluciones con el método de generación de las mismas.

Scatter Search consta básicamente de cinco procedimientos o métodos, estos cinco métodos son comunes a cualquier problema que se resuelva con Búsqueda dispersa pero aquí ya los explicaremos adaptándolo para resolver el problema del viajante de comercio:

Método de Diversificación. Es la fase inicial del heurístico y trata de crear un conjunto *P* con posibles soluciones, esto es, crear la población inicial. Estas soluciones son creadas con un heurístico de construcción. El número de individuos generados o tamaño de la población inicial es un parámetro de entrada pero típicamente suele tener un tamaño de 10. Este conjunto recibe el nombre de *conjunto de referencia (Reference Set)*. Otra opción es separar el concepto del conjunto P y el conjunto de referencia. En este caso, el conjunto P suele tener un tamaño de 100 y de éste se extrae el conjunto de referencia con un cardinal de 10. Evidentemente el tamaño óptimo del conjunto P y conjunto de referencia es un tema de discusión.

Método de Mejora. Esta fase aplica a todos los individuos que van a formar parte de la población o conjunto de referencia un heurístico de mejora que suele ser normalmente un algoritmo de búsqueda local. También suele aplicarse esta fase a las soluciones que se obtienen del método de combinación. La intención de esta fase es mejorar cada individuo de la población para promocionar su supervivencia y incrementar la calidad del resultado global.

Método de actualización del conjunto de referencia. Las soluciones en el conjunto de referencia se encuentran ordenadas de mejor a peor respecto a su calidad. Así que, la adición de nuevos miembros a este conjunto se debe tener en cuenta el valor de su camino para conocer, primero, su lugar en el conjunto y,

segundo, no eliminar por error a buenas soluciones que ya pertenecían al conjunto de referencia. Como este conjunto tiene un cardinal establecido como parámetro de entrada y no puede ser modificado, cada vez que una nueva solución es añadida, otra tendrá que abandonar dicho conjunto, la cual suele ser la peor de todas ellas.

La actualización de este conjunto se produce a través del método de combinación que origina nuevas soluciones que reemplazarán a otras en el conjunto de referencia si su calidad se lo permite y se mantienen la premisas expuestas en el párrafo anterior.

Según de la implementación de la que se trate no sólo la combinación es la única fuente de nuevas aportaciones al conjunto de referencia, ni la fase de actualización de éste atiende estrictamente a criterios cualitativos sino que puede considerarse la diversidad del mismo.

Método de generación de descendencia. Es un método para generar subconjuntos del conjunto de referencia a los que se le aplicará el método de combinación. Al subconjunto lo llamamos en la implementación realizada para esta biblioteca, conjunto de descendencia. Este método es, quizás, el que más caracteriza a Scatter Search que trata de examinar exhaustivamente todas la combinaciones del conjunto de referencia. Esta fase especifica la forma en que se va a explorar el conjunto de referencia para aplicarles el método de combinación y crear el conjunto de descendencia.

Según esta fase se ha implementado en este proyecto la técnica de Búsqueda dispersa el conjunto de referencia es recorrido por pares, esto es, se tiene en cuenta a todas la pareja posibles que se pueden realizar con los miembros de este conjunto. A cada una de estas parejas que van surgiendo se les aplica el método de combinación, esto es, Path Relinking, y el resultado se va añadiendo al conjunto descendencia.

Una mejora en el tiempo de ejecución, que también se encuentra presente en el algoritmo Scatter Search codificado para este proyecto, es marcar a los miembros del conjunto de referencia para saber su longevidad. De esta manera, a la hora de realizar parejas se pueden descartar las parejas que ya se han examinado con anterioridad y evitar la repetición de cálculos innecesaros, pues no repercutirá en el resultado del método debido a que Path Relinking es marcadamente determinista y una misma pareja siempre dará una misma respuesta.

Método de combinación. Hemos hablado de el durante todo el apartado. Es el método o procedimiento por el que los subconjuntos del conjunto de referencia se “mezclan” para generar nuevas soluciones. En el caso de nuestra implementación para el problema del viajante de comercio, estos subconjuntos son las parejas que se extraen del método anterior, método de generación de descendencia.

Como hemos introducido al principio del apartado para Búsqueda dispersa, el método de combinación puede ser elegido de entre muchas técnicas. Sin embargo, sobretodo recientemente, la tendencia entre los investigadores y los mejores resultados para optimización combinatoria se están produciendo con el heurístico Path Relinking. En especial, porque permite una independencia entre los métodos de construcción de nuevos individuos, así como, del método de mejora.

Conociendo ya los métodos de los que consta Scatter Search vamos a describir básicamente el funcionamiento del heurístico. Posteriormente presentaremos una ilustación y un diagrama que apoyará esta descripción.

Inicialmente el algoritmo aplica el método de diversificación, esto es, crea una población inicial. Esta población serán los primeros caminos que integrarán el conjunto de referencia. Los caminos del conjunto de referencia en todo momento se encontrarán ordenados de menor a mayor respecto al valor de la longitud del camino.

Estos primeros caminos del método de diversificación son creados por un heurístico de construcción para el TSP que es tomado como parámetro de entrada. De esta manera, fácilmente puede ser cambiado el método de creación de nuevas soluciones.

A cada nuevo camino creado, antes de ser introducido en el conjunto de referencia, es mejorado a través del método de mejora. La fase de mejora consta de la aplicación de un heurístico de mejora para el TSP. Igualmente, este heurístico es un parámetro de entrada para el Scatter Search que nos posibilitará elegir cómodamente cual emplear.

Una vez que tenemos el primer conjunto de referencia, es decir, la primera generación. Iniciamos un bucle en el cual cada iteración consta del siguiente procedimiento: se obtiene un conjunto de descendencia mediante la aplicación del método de combinación al conjunto de referencia, esto es, la fase de creación del conjunto de descendencia.

También, antes de introducir a los nuevos miembros del conjunto de descendencia, se les aplica el método de mejora.

Una vez finalizada la obtención del conjunto de descendencia, se crea al i-ésima generación de individuos, esto es, se aplica el método de actualización del conjunto de referencia.

Esta iteración es repetida en el bucle en el que se halla anidada. Cuando el bucle termina. El camino solución a la instancia del TSP que esté ejecutando Scatter Search se encuentra en el primer lugar del conjunto de referencia.

El fin del bucle coincide con el criterio de parada del algoritmo. Scatter Search finaliza con normalidad cuando no se generen nuevos caminos o soluciones. Otra forma de ver esta condición es que el conjunto descendencia sea vacío.

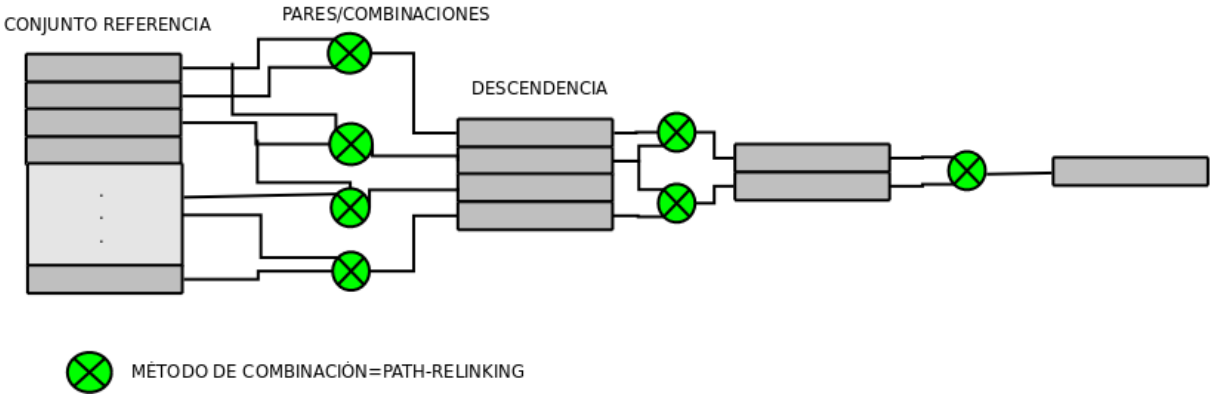


Fig. 3.10. Evolución de la descendencia en Scatter Search.

En la figura 4.10. Estamos viendo como evoluciona la generación de nuevas soluciones o caminos. Podemos observar que en cada iteración el número de nuevos individuos va decreciendo. Esto es debido a que el tamaño de la población es siempre la misma; esto es, el cardinal del conjunto de referencia no varía. Entonces el número de nuevas parejas que se pueden escoger del conjunto de referencia va disminuyendo conforme avanza la ejecución del algoritmo. Recordemos que la descendencia -nuevos caminos- que observamos en la figura 3.9. se incorpora al conjunto de referencia mediante el método de actualización de dicho conjunto, aunque esta figura no lo refleja porque no es una traza del algoritmo; tan sólo es una representación de la evolución en el número de nuevos caminos que se crean.

Gracias a que cada vez se incorporan menos camino en cada iteración, nos asegura que el algoritmo acabará.

Repetimos de nuevo, que las parejas que se constituyen son todas las parejas posibles que se pueden formar entre todos los individuos del conjunto de referencia. Sin embargo, esta estrategia nos puede llevar a repetir cálculos innecesarios porque se repetirían algunas parejas entre algunos individuos supervivientes de generaciones anteriores. Es por esto, como ya adelantamos antes, que se ha introducido una mejora en el algoritmo para evitar realizar el método de combinación (Path Relinking) a parejas que ya han pasado por ese proceso con anterioridad y cuyos resultados se volverían a repetir debido a que Path Relinking es un heurístico determinista. De este modo, cada camino o solución cuenta con un atributo de longevidad que determina si se acaba de incorporar al conjunto de referencia o si ya estaba ahí de iteraciones anteriores. Así, cuando se esté explorando para elegir parejas, si escogemos a dos caminos cuyos atributos de longevidad para ambos nos indican que ya vienen de generaciones anteriores los dos, entonces saltaremos esta pareja y nos ahorraremos el tiempo de ejecución que duraría el método de combinación para ellos dos. Por tanto, sólo aplicaremos Path Relinking a aquellas parejas cuyos miembros sean los dos o al menos uno de ellos nuevo en el conjunto de referencia.

El funcionamiento general del algoritmo lo podemos representar en el siguiente diagrama:

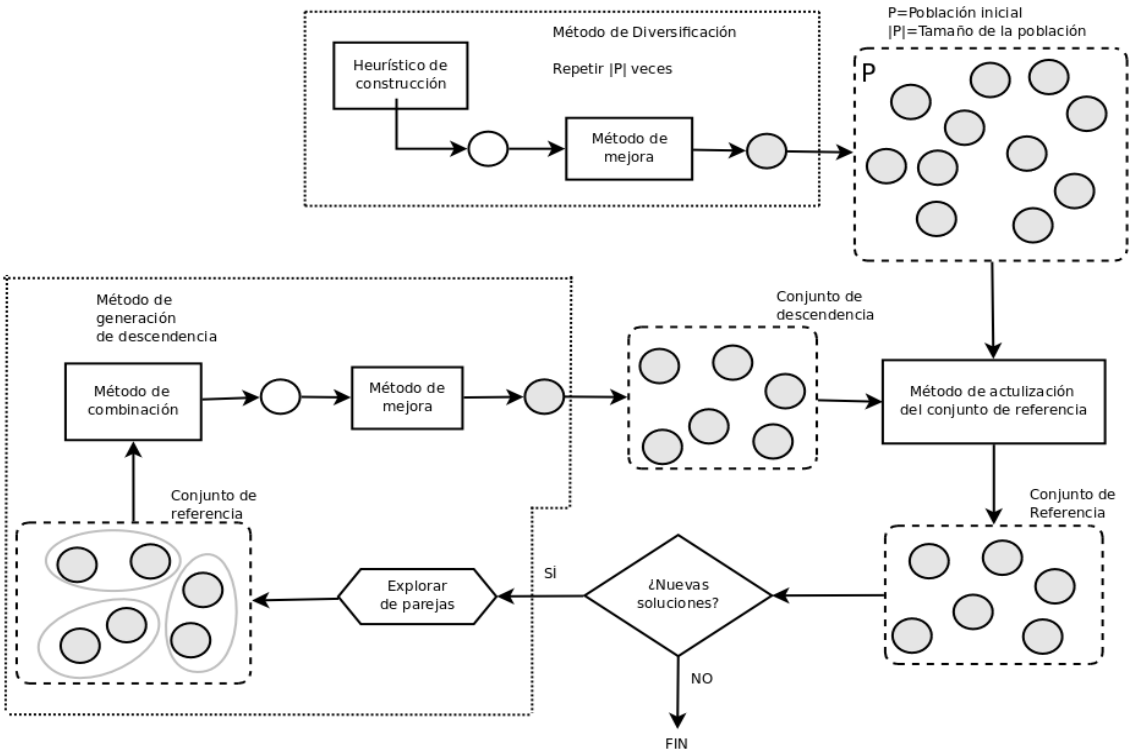


Fig. 4.11. Diagrama de flujo de Scatter Search.

Atendiendo al diagrama de la figura 4.11, el algoritmo de Scatter Search en pseudocódigo podría ser como este:

```
TOUR SCATTER_SEARCH (C:HEURISTICO DE CONSTRUCCIÓN
                     M:HEURÍSTICO DE MEJORA
                     POBLACION:ENTERO)
VARIABLES
     REFSET TIPO ÁRBOL_HEAP
     DESCENDENCIA TIPO ÁRBOL_HEAP
     NUEVA_SOLUCION TIPO LÓGICO
INICIO

// CREAR CONJUNTO REFERENCIA. DIVERSIFICACIÓN
FOR i=1 HASTA POBLACION HACER
     CAMINO=C.GENERATE(); //CREA UN CAMINO
     MCAMINO=M.IMPROVE(CAMINO); //MEJORA EL CAMINO
     REFSET.AÑADIR(MCANIMO);
FIN_FOR

NUEVA_SOLUCION=TRUE;
MIENTRAS (NUEVA_SOLUCIÓN) HACER
     NUEVA_SOLUCIÓN=FALSE;
     //CREAR DESCENDENCIA
     MIENTRAS HAYAPAREJAS(REFSET) HACER
          CAMINO1,CAMINO2=SELECCIONAR UNA PAREJA
          //MÉTODO DE COMBINACIÓN
          INDIVIDUO=PATH_RELINKING(CAMINO1,CAMINO2)
          MINDIVIDUO=M.IMPROVE(INDIVIDUO);
          DESCENDENCIA.AÑADIR(MINDIVIDUO);
     FIN_MIENTRAS
     //MÉTODO DE ACTUALIZACIÓN CONJUNTO REFERENCIA
     MIENTRAS NOT DESCENDENCIA.VACIO() HACER
          X=DESCENDENCIA.DARPRIMERO();
          DESCENDENCIA.ELIMINAR(X);
          SI VALOR(X) < VALOR(REFSET.DARPRIMERO()) ENTOC
               REFSET.AÑADIR(X);
               REFSET.ELIMINARULTIMO();
               NUEVA_SOLUCION=TRUE;
          FIN_SI
     FIN_MIENTRAS
FIN_MIENTRAS
SOLUCIÓN=REFSET.DARPRIMERO();
DEVOLVER (SOLUCIÓN);
FIN SCATTER_SEARCH
```

En cuanto al pseudocódigo sólo vamos a justificar el uso de árboles balanceados para representar la estructura de datos del conjunto de referencia y conjunto de descendencia. Recordemos Scatter Search requiere que los elementos del conjunto de referencia se encuentren ordenados de menor a mayor respecto al valor de la longitud del camino de éstos. Evidentemente se puede conseguir este objetivo con distintas implementaciones, sin embargo hemos optado por el uso de este tipo de árboles por las siguientes razones:

- Se ha pensado que sería mucho más claro a la hora de ver el código.
- Si se hubiera implementado con otros tipos de datos, como el tipo conjunto, etc; entonces habría que haber implementado una función o procedimiento que ordenara los elementos dentro de dicha estructura de datos. Si por algún motivo se cambiara el tipo de datos, necesariamente se modificaría dicha función pues se encuentra íntimamente ligada con los aspectos del tipo de dato elegido; dificultando cualquier modificación posterior.
- El uso de una rutina de ordenación hubiera consumido numerosos recursos pues tienen un orden de complejidad superior al de los árboles balanceados y a esto hay que sumar, además, que la llamada a la función de ordenación se realizaría varias veces durante una misma iteración del algoritmo y una por cada conjunto (de referencia y descendencia).
- Los árboles balanceados nos permiten tener los conjuntos ordenados en cualquier momento y cualquier punto del algoritmo sin un coste computacional extra.
- Los árboles balanceados tienen un orden de complejidad para las operaciones de añadir, eliminar y consultar de O(log n).
- Al usar JAVA para implementar el trabajo nos permite el uso de sus propias bibliotecas y usar la implementación de los árboles balanceados que ofrece este lenguaje. Así obtenemos un implementación de estos árboles con todas las características de JAVA, en cuanto a encapsulado, tratamiento de errores, etc.

  De igual manera, nos garantizamos disfrutar de las últimas capacidades y mejoras en la eficiencia que puedan añadir a los árboles balanceados desde la biblioteca de JAVA en cada actualización de la máquina virtual de este lenguaje.

El método “generate” en la biblioteca implementada para este proyecto, además del común para todos los heurísticos de construcción sin argumentos, tiene la siguiente cabecera.

```
public void generate(TspConstructionHeuristic
builder,TspImprovementHeuristic filter,int population)
```

Mejoras en el heurístico.

Hasta el momento podemos decir que hemos presentado una versión de Scatter Search básica o estándar. Esta versión básica implementada en la biblioteca de este proyecto de esta técnica funciona bien y es rápida. Sus resultados, que podremos ver en el siguiente capítulo, están por encima de otros heurísticos.

Fundamentalmente esta versión sólo atiende a mejorar los aspecto cualitativos de la población.

Scatter Search es un método cuyos principios fueron establecidos hace treinta años pero su desarrollo y aplicación son recientes y numerosos autores proponen nuevas características a esta técnica para mejorar su comportamiento. Las contribuciones son relativas a muchos aspectos del método: estudio del tamaño óptimo del conjunto de referencia, uso de memoria en el método de diversificación restringiendo la creación al azar de nuevos individuos con estructuras similares, seleccionar a los individuos a los que se aplica el método de mejora, estudio de actualizaciones estáticas o dinámicas del conjunto de referencia, etc

Las mejoras introducidas en nuestra implementación para Scatter Search están dirigidas a aumentar la diversidad de la población. Para ello se han introducido los siguientes aspectos:

- Se ha añadido un nuevo parámetro de entrada al método “generate” que indicará al algoritmo la cantidad máxima de nuevas soluciones que se quieran generar. Este valor permite modular la diversidad de la población pues cuanto mayor sea, mayor será el número de individuos que serán necesarios introducir en la población para alcanzar nuevas soluciones. Como efecto colateral este valor modificará el tiempo de ejecución del algoritmo pues a más soluciones más tiempo es el empleado por Scatter Search para encontrar una solución. Típicamente este valor suele ser el número de ciudades de la instancia multiplicada por 10.

- Viendo el criterio de parada del método básico observamos que el heurístico termina cuando no encuentra más soluciones. Esto ocurre porque se parte de una población inicial que no contempla la adición de nuevos miembros que no mantengan una relación con sus progenitores, esto es, todos los nuevos caminos que se añaden al conjunto de referencia

después del método de diversificación invocado al principio de Scatter Search son producto del método de combinación.

La idea de estas mejoras es añadir nuevos caminos al conjunto de referencia para “enriquecer” a la población con nuevos miembros con nuevas características estructurales.

Para ello se introduce un nuevo método a Scatter Search que se encargará de esta tarea, dicha fase se le llama **método de reinicio del conjunto de referencia**.

Sin tener que modificar en demasía el flujo del método básico podemos aumentar la diversidad de este conjunto. Una vez que el algoritmo finaliza porque no encuentra más soluciones, se produce un rearme o reinicio del algoritmo a través de la nueva fase introducida, para comenzar de nuevo otra ejecución con nuevos individuos. Este nuevo método mantiene un porcentaje de las mejores soluciones encontradas hasta ese momento, típicamente un 10% de los miembros del conjunto de referencia; el resto de caminos son eliminados del conjunto de referencia. Los espacios que quedan disponibles entonces son ocupados por nuevos individuos que se obtienen de volver a invocar al método de diversificación.

Estas modificaciones implica cambiar el criterio de parada que pasa a ser que el algoritmo termina cuando el número de nuevas soluciones encontradas en cada iteración no sobrepase el número máximo de soluciones que son tomados como parámetro de entrada en el método “generate”, cuya cabecera queda de la siguiente manera:

```
public void generate(TspConstructionHeuristic
builder,TspImprovementHeuristic filter,int population,int maxSolutions)
```

El objetivo fundamental de estas mejoras introducidas es evitar que el algoritmo quedara encajonado en un óptimo local debido al empleo de una misma población durante toda la ejecución del algoritmo. Estas nuevas características añadidas permitirán explorar nuevos lugares del espacio de soluciones que no estaban dentro de la población inicial, pues rompe el carácter determinista del algoritmo, introduciendo una cierta aleatoriedad.

Aunque veremos los resultados en el capítulo cuarto podemos adelantar que los resultados son muy satisfactorios, encontrado incluso el óptimo a varias instancias del TS;, pese a aumentar los tiempos de ejecución.

El nuevo diagrama de flujo queda de la siguiente manera:

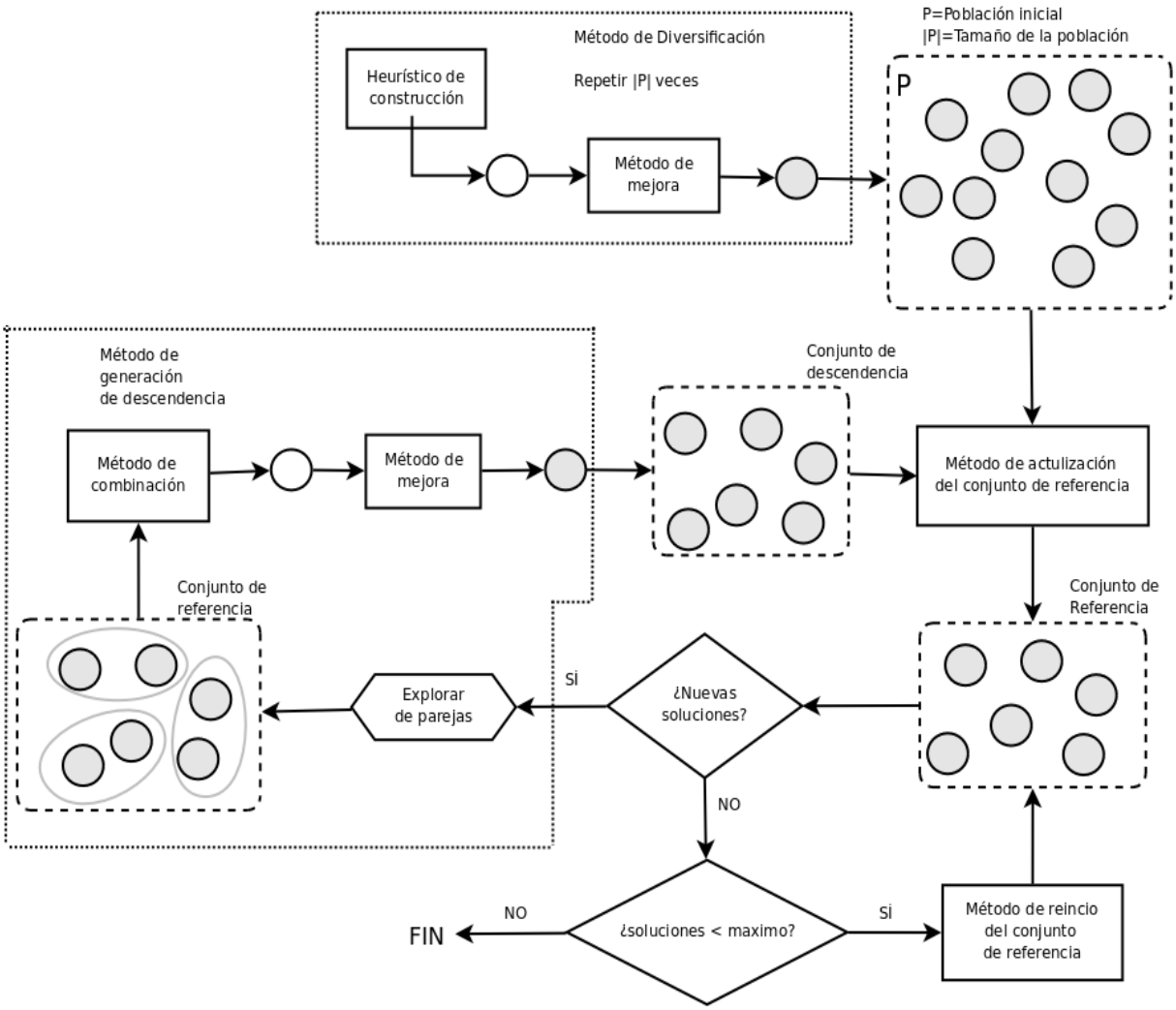


Fig. 4.12. Diagrama de flujo de Scatter Search con las nuevas mejoras.

El método de reinicio del conjunto de referencia es una clase independiente y está fuera de la propia implementación de Scatter Search que actúa como protocolo, mientras el método de reinicio del conjunto de referencia actúa como un controlador de PeerSim.

Este diseño nos da la versatilidad para realizar simulaciones de Scatter Search en su versión básica (sin el método de reinicio) o la versión mejorada añadiendo el método de reinicio del conjunto de referencia; con tan sólo incluir o no en el fichero de configuración de la simulación dicho controlador.

4.4.- HEURÍSTICOS DE MEJORA.

4.4.1.- CLASES PADRES.

Esta clase es la clase padre o de la que hereda los heurísticos de mejoras implementados para el presente trabajo. Es análoga con la clase padre de los heurísticos de construcción, por tanto, es una clase abstracta cuyo cometido es ofrecer una interfaz común a los usuarios de los heurísticos de mejoras para facilitar el uso de estos métodos.

Para los heurísticos de construcción la clase padre era realmente compleja pues ha de considerar una instancia para el tipo de problema del viajante de comercio que se pretendía resolver. Ofrecer estructuras de datos para albergar dicha instancia y los caminos que genere así como sus atributos asociados como valor de la longitud de los caminos, …, además de métodos para el manejo de instancias y manejo de los caminos y sus atributos.

Sin embargo, para la clase padre los heurísticos de mejoras la situación es más fácil; en parte, porque su clase hermana ha hecho gran parte del trabajo. Así, esta clase no necesita mantener una instancia para el TSP porque ya se la encuentra presente, y de la misma manera para los caminos.

Esta clase abstracta para los heurísticos de mejora recibe el nombre de *TspImprovementHeuristic*.

- Constructor.

Sólo cuenta con un único constructor que recibe como parámetro de entrada la instancia del TSP que se encuentre en procesamiento.

- Métodos.

Un método abstracto que define la cabecera común de llamada a los heurísticos de mejora. Este método, efectivamente, será igual para todos y recibe como parámetro de entrada el camino que se quiere mejorar. Este parámetro es pasado por referencia con la intención de devolver por la misma variable del camino dado como argumento de entrada, el resultado del método de mejora que es consecuencia otro camino. La cabecera es la siguiente:

```
public abstract void improve(int t[ ])
```

Como esta clase es muy pequeña expondremos aquí su código en su totalidad para verla íntegramente.

```
public abstract class TspImprovementHeuristic {

            protected TSP instance;

    /** Constructor */
    public TspImprovementHeuristic(TSP g) {
        instance=g;
    }

    public abstract void improve(int t[]);

}
```

Como hemos hecho con los heurísticos de construcción, lo explicado hasta el momento es para ejecuciones del heurístico en un único computador. El fin de estas implementaciones en este trabajo fin de grado es para su uso auxiliar en otras clases que precisen de un heurístico de mejora.

Si bien, la clase padre para los heurísticos de mejora carece de otros métodos, no ocurre así a su adaptación para ambientes P2P donde será usada como protocolo dentro de los nodos de la red.

El cambio más importante es que el método abstracto *improve(int t[]),* cambia de nombre a *mutate(int t[])* para ser exactos en su uso. En una ejecución independiente y única el método de mejora siempre va a devolver un camino mejor. En un entorno P2P donde existe comunicaciones constantes con otros nodos el tour de referencia es cambiante y, por tanto, no podemos hablar de una mejora absoluta porque puede haber cambiado. Así el método se llamará *mutate* porque realizará un cambio en el tour de referencia del instante *t* que no podemos asegurar que al finalizar la ejecución del método su resultado sea una mejora, ya en el instante $t+\Delta_t$.

Existe otro motivo para el cambio de nombre, ya que, para unos heurísticos sencillos cuyo nivel de mejora por si sólo es mínimo, se ha querido probar una estrategia de ejecutarlo en un entorno distribuido como es una red P2P y que generen múltiples mutaciones de un camino dado, donde habrá una especie de selección donde la red P2P se va quedando con los mejores.

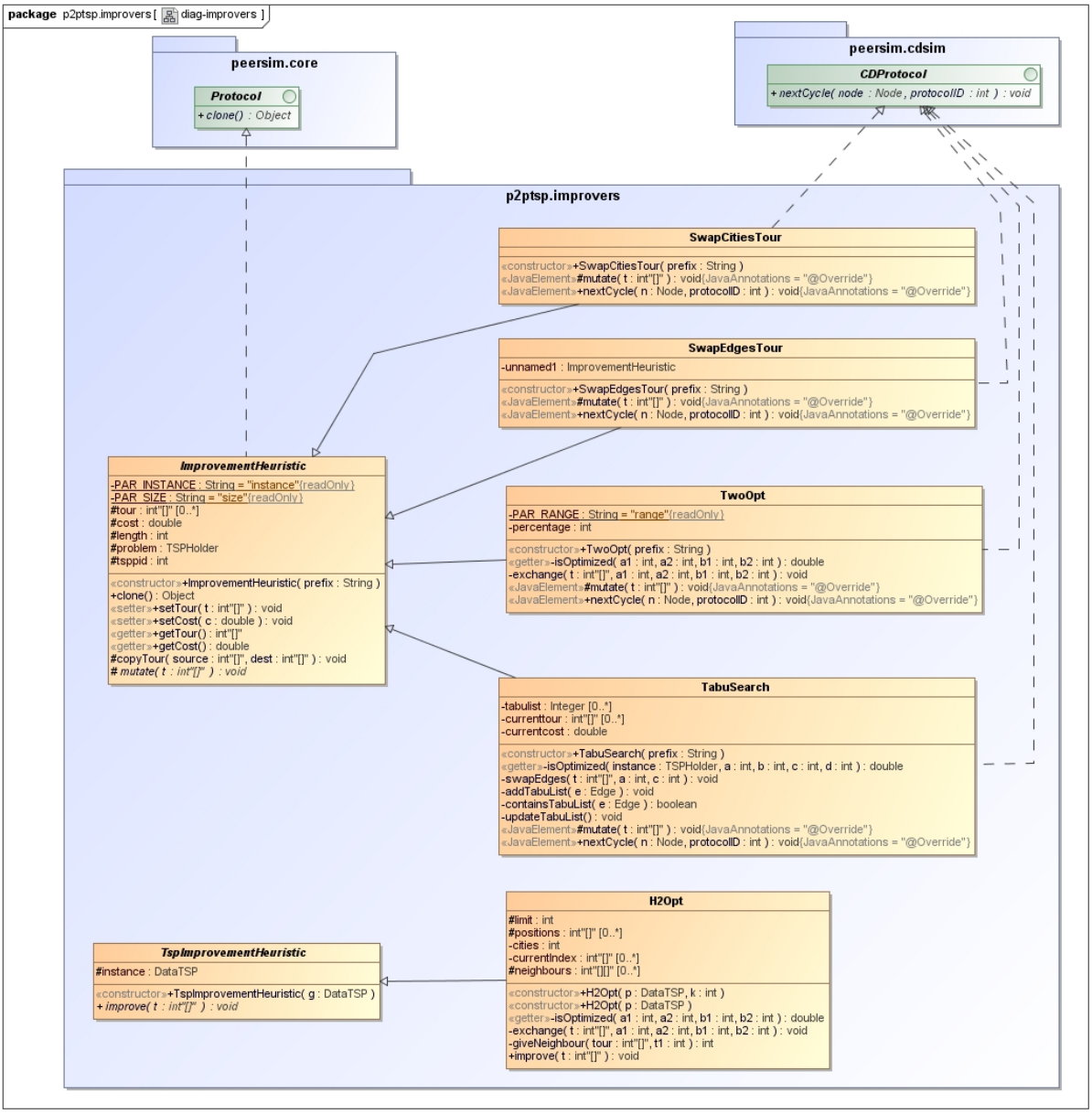


Fig. 4.13. Diagrama UML para el paquete de heurísticos de mejora.

4.4.2.- NEIGHBOUR IMPROVEMENT.

Este es un heurístico muy sencillo e ingenuo en su planteamiento pero que nos sirve bien para dar buena muestra sobre el funcionamiento de este tipo de técnicas vinculadas a la búsqueda local.

Como su nombre indica en inglés, este algoritmo trata de mejorar el camino dado observando a los vecinos de cada ciudad e intercambiándose por estos si se produce una reducción en el valor total de la longitud del camino. En cada iteración se visita a una ciudad y se compara con su vecino. Esto ocurre sucesivamente hasta que se hayan examinado a todas la ciudades.

Podemos ver más claramente esta operación viendo en la siguiente figura 4.14 un ejemplo de una iteración de esta técnica. Encontrándose en la primera iteración del algoritmo, cuando i=1, y asumiendo que la primera ciudad del camino recibe como número de orden el cero. Tenemos que el primer arco está compuesto por las ciudades que se encuentran en la posición i, es decir, la segunda ciudad (i=1) y la ciudad de la posición i-1, la primera de ellas.

Entonces el arco vecino al arco actual, será las ciudades que se encuentren en las posiciones (i+1) y (i+2), tercera y cuarta ciudad respectivamente.

Fijándonos en el camino de la figura 4.14, tenemos que estos arcos son (a,b) como arco actual y a (c,d) como arco vecino.

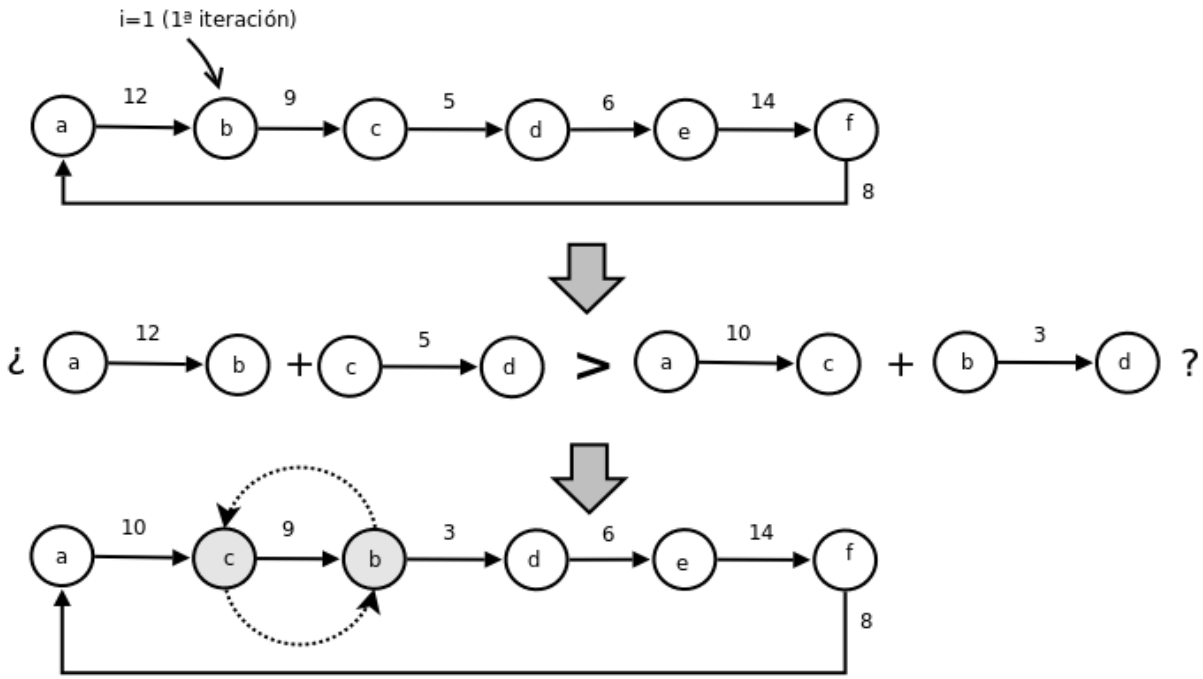


Fig. 4.14. Iteración del heurístico Neighbour Improvement.

El algoritmo se pregunta si la longitud de estos dos arcos, d(a,b)+d(c,d), es mayor que si intercambiamos a las ciudades de la siguiente manera donde su longitud sería, d(a,c)+d(b,d). En caso afirmativo, se procederá al intercambio que tendrá como consecuencia inmediata la disminución de la longitud total del camino completo.

En caso contrario, no se produce el intercambio para no empeorar el camino y se pasa a la siguiente iteración, incrementando en una unidad el valor de la variable i.

Cuando i alcanza el valor del número total de ciudades de la instancia, esto es, se encuentra en la última ciudad del camino, el método finaliza.

Vemos que este método de búsqueda local explora un radio muy reducido del espacio de soluciones, pues considera sólo a su vecino más próximo. Como ventaja ofrece que el algoritmo convergerá rápidamente pero tiene el gran riesgo de quedar “encajonado” en un óptimo local sin posibilidad de realizar grandes mejoras o ninguna.

Este algoritmo de mejora del vecino realmente no se encuentra implementado de esta forma en este trabajo fin de grado. Lo hemos descrito para que quede patente cual es la estrategia de estos heurísticos cuando son ejecutados independientemente y en solitario. Y podemos contrastar el cambio que se produce a una adaptación en un entorno distribuido donde se multiplican el número de ejecuciones del mismo algoritmo en el mismo instante de tiempo y obtener tantos resultados diferentes y simultáneos como número de componentes tenga la red P2P. De esta manera, se quiere saltar la restricción comentada de quedar encallado en un óptimo local cuando existe una sola línea de ejecución.

Intercambio de ciudades

Este algoritmo (SwapCitiesTour) escoge dos ciudades dentro del tour aleatoriamente y las intercambia dentro del mismo camino. El movimiento muy similar al de mejora del vecino, se realiza sin ningún tipo de comprobación de que sea beneficioso. Se obtiene un tour distinto al original, cuando el nodo que lo ha generado comprueba si es un candidato mejor de que el ya tenía. Sí es así, actualiza su información almacenado en el nuevo tour y el valor del mismo; en caso contrario, es desechado.

Otro protocolo del mismo nodo encargado de comunicarse con otros elementos de la red, difunde su información y recibe otra, donde podrá decidir si mantener el tour que ya tiene o ha encontrado por la red una alternativa mejor.

Intercambio de aristas

Bajo el mismo concepto que el anterior, con la única diferencia que en lugar de intercambiarse dos ciudades en el tour, se toman al azar dos aristas dentro del camino. El resto de la operatoria es igual al algoritmo de intercambio de ciudades.

4.4.3.- 2-OPTIMAL (2-OPT).

Este heurístico de mejora, perteneciente también a la familia de métodos de búsqueda local, fue desarrollado especialmente para el problema del viajante de comercio en 1958 por G. A. Croes observando que la eliminación de los ejes cruzados de un camino para una instancia del problema del viajante de comercio se obtiene otro camino de menor longitud. Esta situación cobra aún más fuerza cuando nos referimos a instancias que se establecen en espacios euclidianos.

Claramente podremos observar el principio sobre el que se basa este heurístico en la siguiente figura.

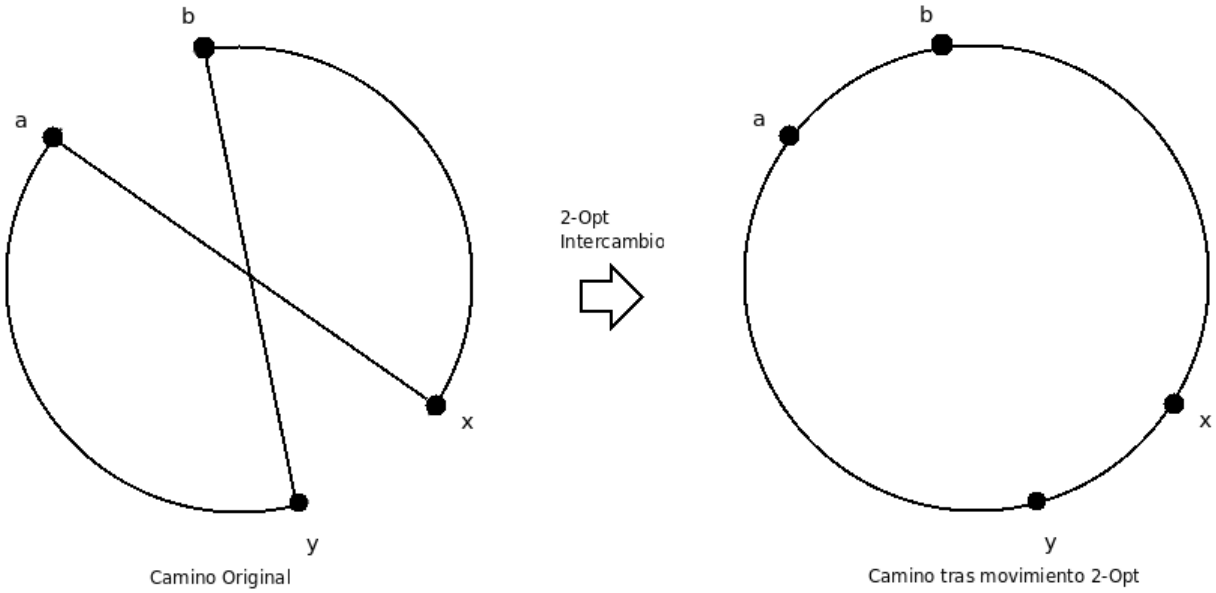


Fig. 4.15. Operación de movimiento 2-Opt.

En la ilustración (figura 4.15) tenemos un ejemplo de lo que se llama *movimiento 2-Opt*. Este movimiento fue descrito antes que el heurístico completo en 1956 por M.M. Flood el cual inspiró al desarrollo de esta técnica.

Si los dibujos de la figura 4.15 los ubicamos dentro de un plano euclidiano, donde el tamaño de las líneas se corresponde con la longitud de éstas, nos percatamos, sin lugar a dudas, que las aristas del camino original (a,x) y (b,y) tienen una longitud mayor que las nuevas que se han incorporado, (a,b) y (x,y).

Por tanto, un movimiento 2-Opt consisten en eliminar las aristas que producen el cruce, en nuestro caso (a,x) y (b,y), y añadir otras dos para que se pueda obtener un ciclo hamiltoniano, pues ha quedado roto tras la eliminación de aristas.

Esta operación es muy simple porque, como podemos comprobar, una vez

eliminadas las aristas que producen el cruce, solamente tenemos una única opción de añadir aristas si queremos volver a obtener un ciclo hamiltoniano. Afortunadamente, esta única opción es la que nos interesa para el movimiento 2-Opt, que es añadir la aristas más cortas que unen a las ciudades y que dan a lugar a un ciclo hamiltoniano; en nuestro caso, es añadir las aristas (a,b) y (x,y).

El heurístico se basa en realizar el movimiento 2-Opt a todos de los cualesquiera pares de aristas que estén dentro de un camino para el TSP dado, si ello produce un beneficio, esto es, se consigue una reducción de la longitud de dicho camino.

El algoritmo que implemente este heurístico tendrá que escoger en cada iteración un par de aristas y comprobar que si realiza una movimiento 2-Opt con ellas, obtiene una mejora del camino. Si es así, entonces ejecuta el movimiento 2-Opt y pasa a la siguiente iteración. En caso contrario, deja el camino como estaba y se va a la siguiente iteración.

El algoritmo para cuando ya se han explorado todas las aristas del camino.

Veamos en pseudo-código la implementación de este método.

```
ALGORITMO 2-OPT (VAR T:TOUR)

VARIABLES
     T' TIPO TOUR;
     I,J TIPO ENTERO;
     M,N TIPO ARISTA;

INICIO
     PARA I=0 HASTA T.LONGITUD() HACER
          M=CREAR_ARISTA(T(I),T(I+1));
          PARA J=I HASTA TOUR.LONGITUD() HACER
               N=CREAR_ARISTA(T(J),T(J+1));
               T'=2OPT_MOVE(M,N);
               SI (T.LONGITUD() > T'.LONGITUD()) ENTONCES
                    T=T';
               EN OTRO CASO
                    T=T;
               FIN_SI
          FIN_PARA
     FIN_PARA

FIN_ALGORITMO;
```

Viendo el pseudocódigo notamos que la complejidad computacional del algoritmo pertenece a $O(n^2)$, existen algunas modificaciones para mejorar su eficiencia. No obstante, a grandes rasgo es un método bastante rápido.

En líneas generales, este heurístico suele producir buenos resultados pero éstos no están garantizados. Además, siempre rompe los cruces del camino dado.

Estas heurísticas de mejoras tienen como gran ventaja su uso combinado con otros tipos de heurísticas con el objetivo de mejorar los resultados; tal y como, hemos podido ver en la implementación de este trabajo como en Scatter Search. Por este motivo, 2-Opt se encuentra implementado tanto adaptado como protocolo para ser usado por un nodo directamente como heurístico simplemente para su uso por Scatter Search.

4.4.4.- BÚSQUEDA TABÚ

La búsqueda tabú (*Tabu Search*, en inglés) es un metaheurístico, diseñado por Fred Glover en 1986, perteneciente a la clase de técnicas de búsqueda local; se usa para la resolución de problemas de optimización combinatoria. Como curiosidad, el mismo artículo donde su creador introduce la búsqueda tabú, acuña el término metaheurística. Posteriores artículos siguen desarrollando este algoritmo. Fred Glover dice de esta metaheurística que “la búsqueda tabú guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local”, ya que, su objetivo es evitar que el algoritmo quede atrapado en un óptimo local, como ocurre en la mayoría de métodos de búsqueda local.

Para el problema del viajante de comercio, este algoritmo se clasifica como un heurístico de mejora, ya que, necesita partir de un tour ya formado.

Búsqueda Tabú explora el espacio de soluciones a través de repetidos movimientos desde una solución inicial a la mejor de sus vecinas tratando de evitar los óptimos locales. El heurístico realiza una búsqueda por entornos en la cual se desplaza en cada iteración a la mejor solución no tabú del vecindario de la solución actual. Los principales atributos de cada solución visitada son almacenados en una lista tabú por un determinado número de iteraciones para evitar que estas soluciones sean revisitadas, es decir, para evitar ciclos en la búsqueda por entornos. Así, un elemento del vecindario de la solución actual es declarado tabú (es decir, es prohibido) si alguno de sus atributos está en la lista tabú.

En general, un método basado en búsqueda tabú requiere de los siguientes elementos:

1. Solución inicial. Como método de mejora para el TSP, la búsqueda debe comenzar desde una solución inicial que podría ser cualquier solución admisible

que satisfaga las restricciones del problema.

2. Movimiento. Un movimiento es un procedimiento aleatorio o determinístico por el que se genera una solución admisible a partir de la solución inicial. Usualmente, este procedimiento es sencillo para el caso de problemas combinatorios, pero mucho más complejo para el caso de problemas de optimización continuos. En nuestro caso se tratará de realizar un simple intercambio de aristas.

3. Vecindad. Dada una solución S, la vecindad N(S) es el conjunto de todas la soluciones admisibles que pueden ser generadas por la ejecución de un movimiento sobre la solución actual S. Este conjunto suele ser numerable para problemas de optimización combinatoria y, en aquellos casos en los que N(S) sea grande, se suele operar con un subconjunto de éste.

4. Lista tabú. Es un mecanismo de memoria adaptativa que trata de evitar que la búsqueda entre en un ciclo o quede atrapada en un óptimo local. Una vez que un movimiento, que genera una nueva solución, es aceptado, su movimiento inverso se añade a la lista tabú y permanece en ésta un número determinado de iteraciones. Si el tamaño de la lista tabú es pequeño, entonces la búsqueda se intensifica en una determinada área del espacio, mientras que si el tamaño de la lista es grande se enfatiza la búsqueda en diferentes regiones del espacio de soluciones.

La implementación que se ha realizado para este trabajo la lista tabú no tiene un tamaño determinado, sino que esta varía en función del tiempo que permanecen sus elementos en ella. Cada elemento de la lista tabú, al ingresar, se le asigna un número que indicará el número de ciclos que se mantendrá incluido en la lista. Una vez superado dicho número de ciclos, se borrará de la lista tabú ese movimiento, que volverá a estar disponible para ser tenido en centa por el algoritmo. El número de ciclos que se le asigna a un elemento al ser añadido a la lista tabú, en nuestra implementación, es un número aleatorio dentro del intervalo siguiente:

$\left[\frac{n_t}{2}, \frac{3\ n_t}{2}\right]$ siendo n el tamaño de la red P2P en el instante t de la simulación.

5. Criterio de parada. En general, la búsqueda termina después de un número determinado de iteraciones, después de un tiempo de computación predefinido o cuando se alcanza un número dado de iteraciones sin mejorar la

mejor solución.

Estos cinco elementos constituyen el método básico que sigue Búsqueda Tabú y que toda implementación debe contener. Además, gracias a la continua investigación a la que está sometido este metaheurístico, existen diversas estrategias que se pueden añadir al método básico y que tienen por objetivo la mejora de su desempeño. Entre otras está incorporar una memoria a largo plazo que permita inferir preferencias, a partir de la información almacenada en ella, como la cantidad de veces que una solución es la mejor o cantidad de veces que un atributo pertenece a una solución generada. Fases de intensificación, que permite concentrar la búsqueda en aquellas zonas más prometedoras. Fase de diversificación, que permite desplazarse hacia zonas no exploradas o, la más común, imponer un criterio de aspiración que admite movimientos tabú si se satisface el criterio de aspiración con idea de cruzar las barreras impuestas en las restricciones tratando de encontrar otras zonas factibles más prometedoras.

A nivel de nodo de la red, en la implementación realizada se ha introducido un criterio de aspiración para elementos de la lista tabú que mejoren el valor de la función objetivo de la mejor solución encontrada hasta ese momento.

A continuación veamos en pseudo-código el algoritmo implementado.

```
ALGORITMO TABU-SEARCH (VAR T:TOUR)

VARIABLES
     T' TIPO TOUR;
     I,J TIPO ENTERO;
     GANANCIA TIPO REAL;
     CANDIDATO TIPO ARISTA;
     LISTA_VECINOS TIPO TABLA DE DISPERSIÓN ORDENADA;

INICIO

     OBTENER INSTANCIA DEL TSP DEL PROTOCOLO
     COPIAR T → T'

     //Explorar el espacio de soluciones
     PARA I=1 HASTA TOTALCIUDADES I+1
          PARA J=I+2 HASTA TOTALCIUDADES J+1
               CANDIDATO=ARISTA(T'[I],T'[I+1]),
                                (T'[J],T'[J+1])
               GANACIA=CALCULAR_GANANCIA (CANDIDATO)
```

```
                LISTA_VECINOS.PUT(GANANCIA,CANDIDATO);
           FIN PARA
      FIN PARA

      //Buscar al mejor vecino
      MIENTRAS (HAYVECINOS) HACER
           MENORGAIN=LISTA_VECINOS.GET_LAST_KEY();
           ACTUALCAND=LISTA_VECINOS.GET(MENORGAIN);
           //Preguntamos si es tabú el candidato
           SI LISTATABU.CONTIENE(ACTUALCAND) ENTONCES
                //Es tabú
                //Preguntamos si criterio de aspiración
                SI (MENORGAIN > 0) Y (T.VALOR >
                          (T'.VALOR-MENORGAIN)) ENTONCES
                     //Aplicar criterio de aspiración
                     HAYVECINOS=FALSO;
                     T'.INTERCAMBIAR(ACTUALCAND);
                     COPIAR T' → T;
                EN OTRO CASO
                     //No se aplica criterio de aspiración
                     HAYVECINOS=VERDADERO;
                     LISTA_VECINOS.REMOVE(MENORGAIN);
                FIN SI
           EN OTRO CASO
                //No es tabú
                HAYVECINOS=FALSO;
                LISTATABU.AÑADIR(ACTUALCAND);
                T'.INTERCAMBIAR(ACTUALCAND);
                COPIAR T' → T;
           FIN SI

           SI T'.VALOR < T.VALOR
                COPIAR T' → T
      FIN MIENTRAS

      LISTATABU.ACTUALIZAR_CADUCIDAD();

FIN TABU-SEARCH
```

La complejidad computacional de la implementación realizada pertenece a $O(n^2)$.

4.5.- PAQUETE VECTOR

Este paquete se concibe para albergar clases que afectan a todos los nodos de la red en un mismo ciclo. Las clases de este paquete tienen funciones distintas entre sí. Pueden ser

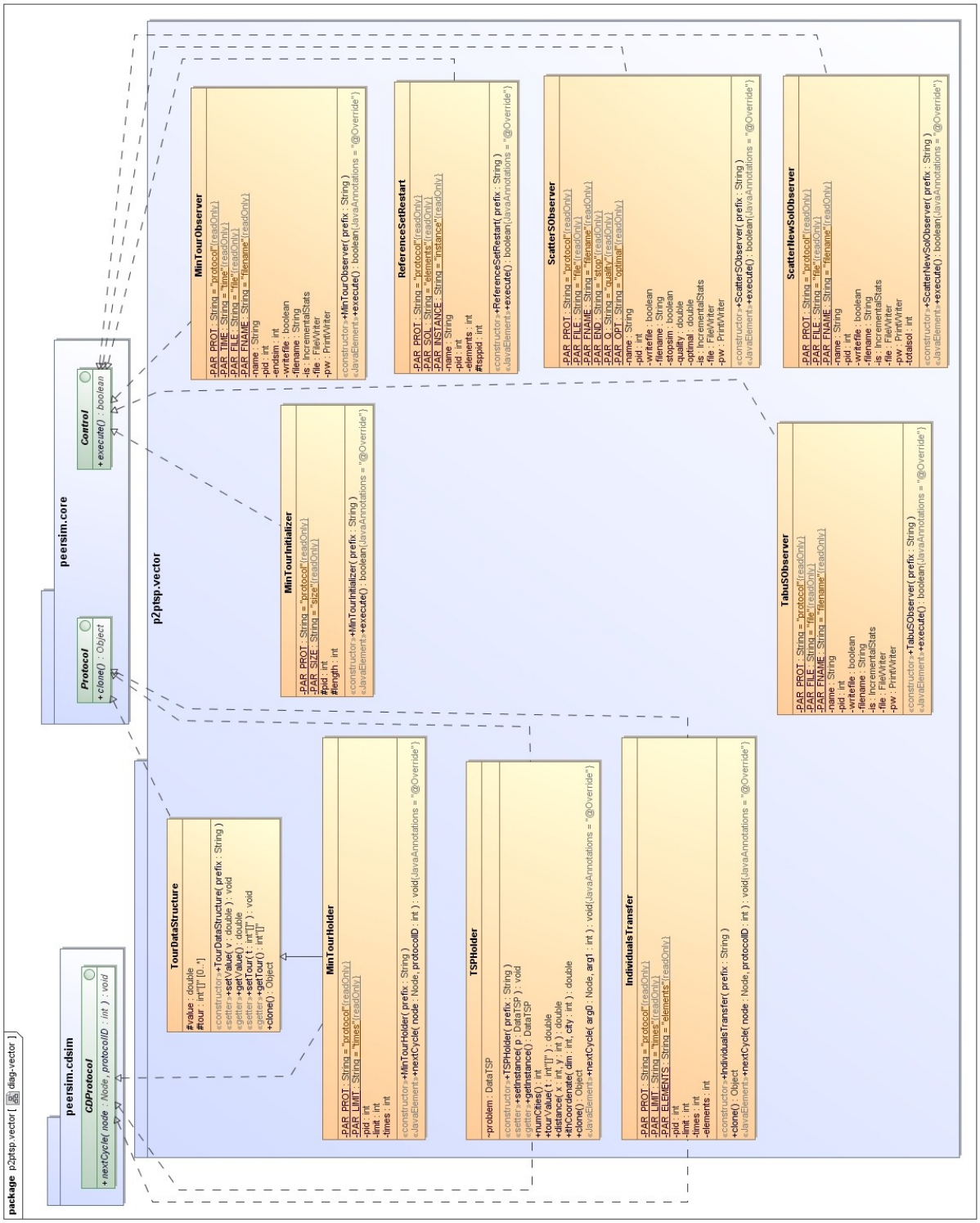


Fig. 4.16. Diagrama UML del paquete vector

observadores, inicializadores, funciones específicas de un metaheurístico o almacenamiento de datos en la simulación; siendo esta última su cometido original según la documentación de PeerSim y cuyo paquete vector también está presente en su implementación. Por este motivo, y porque la representación de la red P2P en PeerSim es un vector, de ahí de llamar al paquete de esta manera.

4.5.1.- ALMACENAMIENTO DE DATOS

Estas clases se implementan como protocolos de los nodos de la red, ya que, su cometido es, primero, almacenar o mantener una estructura de datos y su contenido durante la simulación y , segundo, que dicha información se encuentre disponible por el nodo durante el tiempo que dure la simulación. Además, estas clases ofrecen métodos para el manejo de los datos que guardan.

Por otro lado, algunos de estos protocolos no sólo cumplen su función de almacenamiento sino que son capaces de establecer comunicaciones con otros nodos de la red para realizar un intercambio de información o de recursos según el propósito de la simulación. En nuestro caso, el objetivo será obtener de un vecino un camino de menor coste.

Vamos a enumerar las clases que componen este paquete con una breve descripción:

- TSPHolder: es un protocolo que mantiene en el nodo la instancia del problema del viajante de comercio que se está utilizando para la simulación. Proporciona métodos para el manejo de instancia y obtención de información de la misma, los cuales son homólogos a los que implementa la clase *DataTSP* del paquete instances.
- TourDataStructure: es una superclase de las que pueden heredar otras que quieran almacenar como dato un camino o tour para el problema del viajante de comercio y su coste o valor del recorrido. Tiene métodos de acceso y consulta tipo setter y getter.
- MinTourHolder: es un protocolo que hereda del anterior (*TourDataStructure*) y es ampliamente utilizado en este trabajo. Su función es almacenar el tour mínimo conocido por el nodo que lo contiene. Este protocolo en un intervalo de tiempo predeterminado, mediante el fichero de configuración, realiza labores de comunicación con otros nodos para intercambiarse los tours y quedarse con el menor de ellos.

4.5.2.- INICIALIZADORES

En este caso sólo tenemos un único representante que se trata de la clase *MinTourInitializer*. Es una clase muy sencilla que trata de dar un valor inicial al protocolo *MinTourHolder* al comienzo de la simulación para que tenga un correcto funcionamiento. Como el objetivo es mantener siempre el tour mínimo, lo cual se obtiene realizando comparaciones en el coste del tour. Esta clase al

inicio de la simulación establece como valor del tour inicial el mayor valor que el lenguaje de programación dota al tipo de dato numérico con el que se declara el coste de un tour. En nuestra implementación el tipo de dato es un *double* donde el lenguaje JAVA define una constante para conseguir el máximo valor que puede alcanzar una variable de tipo *double*.

4.5.3.- OBSERVADORES

Dentro de los objetos tipo Control de PeerSim, los observadores son los encargados de tomar información de la simulación en un instante determinado de la misma y realizar con ellas distintas operaciones con el fin de obtener estadísticas, normalización de datos, volcado de datos, etc.

Para todos los heurísticos y/o experimentos que se han desarrollado en este trabajo, se ha implementado un observador. Su función fundamental ha sido la recogida de los datos de interés y su volcado a un fichero para ser tratado mediante programas como Matlab. En algunos casos, se han establecido condiciones que en función de dichos datos provoca la finalización de la simulación.

Enumeramos los observadores de este paquete:

- MinTourObserver: destinado a las heurísticas básicas (el vecino más cercano, 2-Opt, etc). En cada ciclo de la simulación, recorre todos los nodos para obtener su camino mínimo. De todos los datos que recoge, muestra por la salida estándar y por fichero sí así se le indica por la configuración, el valor del camino mínimo de dicho ciclo, el máximo del mismo y la media de todos ellos. Además del tamaño de la red en el ciclo en cuestión. También, mediante el fichero de configuración, se le puede indicar qué ciclo se desea terminar la simulación.
- ScatterSObserver: orientado para su uso con el heurístico Scatter Search es muy similar al anterior observador, pues realiza la misma recopilación de información sobre el coste de los caminos en cada ciclo de la simulación y su volcado por la salida estándar y fichero. En este caso, y como este observador está orientado a valor de la solución, se puede parametrizar a través del fichero de configuración, la finalización del experimento en función de alcanzar un determinado porcentaje de calidad del tour. Por supuesto, se necesita conocer e informar del valor del óptimo para la instancia a tratar.

- ScatterNewSolObserver: Al igual que los anteriores recopila información y la presenta al usuario por los mismos medios. Este observador, en lugar capturar el valor de los tours en los nodos, extrae otro parámetro que también es muy importante en un heurístico poblacional como es Scatter Search; como son la cantidad de nuevos individuos que se generan en cada ciclo de la simulación y formarán parte de la población.
- TabuSObserver: Este observador es homólogo a los demás pero dirigido al protocolo que implementa el mentaheurístico de Búsqueda Tabú. La información que trata es respecto al valor de los caminos.

4.5.4.- OTROS

- ReferenceSetRestart: Scatter Search se ha implementado en este trabajo fin de grado en dos versiones. Una estándar y otra versión mejorada. Recordemos que la versión mejorada suponía realizar un reinicio del conjunto de referencia para diversificar la población. Con la finalidad de facilitar la conmutación entre una versión y otra en la realización de experimentos con este simulador, hemos aprovechado la arquitectura de PeerSim y se ha implementado como objeto de tipo *Control*, este controlador *ReferenceSetRestart* que permitirá el uso de la versión mejora o estándar según se añada a la simulación o no se añada a la misma. Es parametrizable el número de elementos que se incorporan al conjunto de referencia.
- IndividualsTransfer: Este protocolo implementa la adaptación a un red P2P del metahurístico Scatter Search. Se encarga de realizar la comunicaciones para contactar con otros nodos de la red y poder intercambiar el recurso más importante en algoritmos poblacionales como son los individuos que formar la población del experimento. El protocolo obtiene del fichero de configuración el intervalo de tiempo en que se va a llevar a cabo el intercambio de información y el número de elementos que se transferirán en ello.

4.6.- PAQUETE DE DINÁMICA DE LA RED

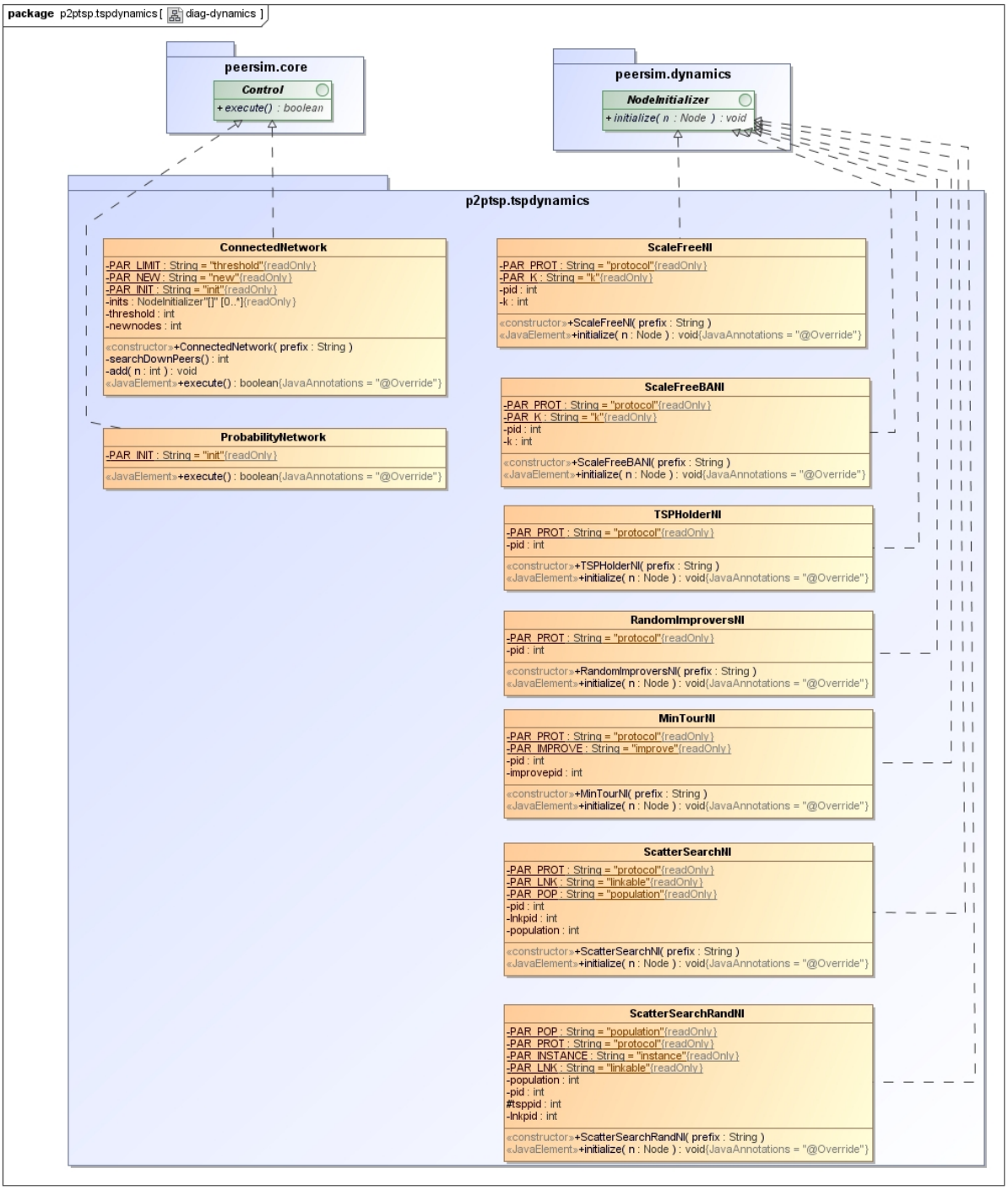


Fig. 4.17. Diagrama UML del paquete tspdynamics

Este paquete se usa para modelar en el simulador PeerSim el dinamismo intrínseco que ya hemos visto en las redes P2P donde aparecen y desaparecen nodos en la misma de forma indeterminada. Este paquete tiene dos variantes, una primera que viene a complementar el conjunto de clases que ya ofrece PeerSim para este propósito y, una segunda variante, que viene a implementar

los procesos que se ejecutan para inicializar los nuevos nodos que se unen a la red P2P una vez esta ya se encuentra en pleno funcionamiento.

Estas dos variantes de las que hemos hablado no son independientes entre sí, sino que la primera – que es la encargada de eliminar y añadir nodos – tiene que establecer un mecanismo para que puedan inicializarse los nuevos nodos que va a añadir durante la simulación. La segunda variante aprovecha dicho mecanismo habilitado por la anterior para realizar la inicialización de los distintos protocolos que forman los nodos de la red.

4.6.1.- MODELADO DEL DINAMISMO DE LA RED

Las clases de este apartado son las que nos hemos referido como primera variante. Su cometido es modelar el dinamismo de la red P2P en el simulador. Existen muchos modelos a seguir, y uno de los más interesantes son los que siguen el paradigma de las redes libre de escala. El propio PeerSim incluye clases en este sentido y se han usado ampliamente en este trabajo.

Este trabajo fin de grado realiza dos aportaciones:

- ProbabilityNetwork. Esta clase modela la entrada y salida de nodos de la red en función de una probabilidad que está basada en el tamaño de la red en el momento de la simulación que es llamada. En el fichero de configuración hay que especificar los parámetros siguientes: tamaño máximo de la red que no se debe sobrepasar, factor de multiplicación para el cálculo de la probabilidad de dinamismo de la red y tantos inicializadores de nodos específicos para cada protocolo como tengan los nodos de la red.
- ConnectedNetwork. Esta clase no es un modelo para la red. Su función es la de un controlador que impide que el tamaño de la red disminuya a un umbral determinado en el fichero de configuración. Como, se ha comentado, el dinamismo de la red es impredecible y se puede dar la situación que el número de nodos caiga de tal forma que se produzca una pérdida de información o que incluso desaparezca la red. Este controlador establece un mecanismo para evitarlo, añadiendo nuevos nodos si así se precisara.

4.6.2.- INICIALIZADORES DE NODOS

Los inicializadores de nodos se encargan de inicializar los protocolos que se están usando en una simulación para los nuevos nodos que se añaden una vez la red P2P se encuentra en funcionamiento. Es decir, estos inicializadores son distintos y no se emplean cuando la simulación empieza a ejecutarse por primera vez.

Para cada protocolo se ha implementado un inicializador de nodo, ya que, deben se específicos para estos.

Estos inicializadores de nodos son especificados en el fichero de configuración como parámetros para los controladores de modelado del dinamismo de la red, de la siguiente manera:

```
control.dnet ProbabilityNetwork
control.dnet.maxsize NETSIZE
control.dnet.multiple 1
control.dnet.init.0 ScaleFreeNI
control.dnet.init.0.protocol lnk
control.dnet.init.0.k K
control.dnet.init.1 TSPHolderNI
control.dnet.init.1.protocol problem
control.dnet.init.2 RandomImproversNI
control.dnet.init.2.protocol improve
control.dnet.init.3 MinTourNI
control.dnet.init.3.protocol min
control.dnet.init.3.improve improve
```

Este ejemplo se corresponde a una simulación real que se ha usado para este trabajo donde se simulan las heurísticas básicas. Los nodos de la red tienen 4 protocolos, el primero con identificador 0 es de tipo *Linkable* y trata la topología de la red, en este caso el nuevo nodo optará por seguir una red libre de escala; el segundo es el protocolo que almacena la instancia del problema del viajante de comercio que se está usando; el tercero es el protocolo para un heurístico de mejora, el inicializador es genérico para todos estos heurísticos de mejora, ya que, no tiene un tour inicial que mejorar, por lo que, la solución pasa para este inicializador en tomar el mejor tour que tenga un nodo activo aleatorio de la red; el último protocolo es el que almacena el mejor tour conocido hasta el momento, el inicializador hace lo mismo para el nuevo nodo que se va añadir tomando el tour que se obtuvo en el protocolo anterior.

4.7.- PAQUETE DE UTILIDADES

Este último paquete contiene unas pocas clases auxiliares a la simulación.

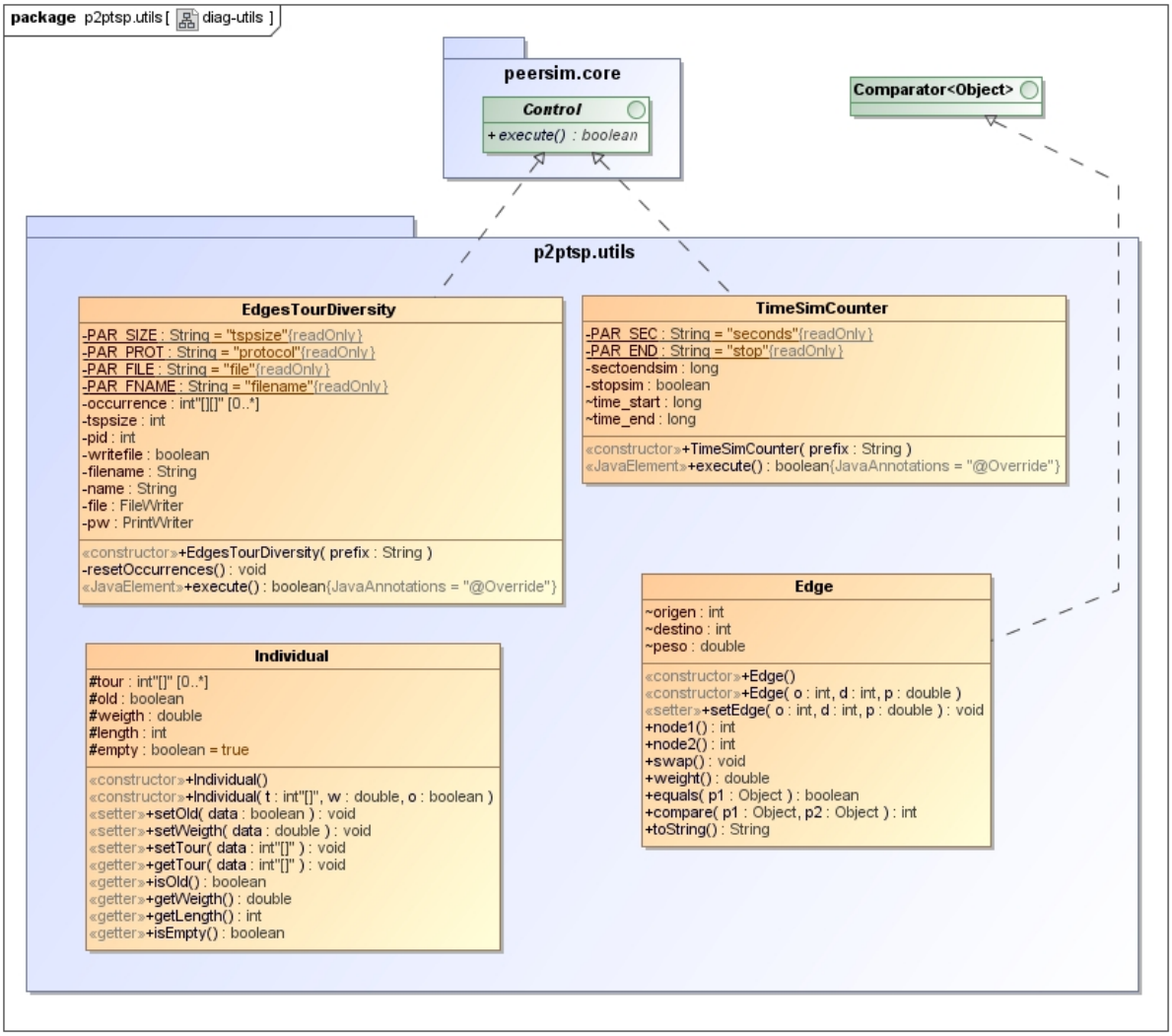


Fig. 4.18. Diagrama UML para el paquete de utilidades

- EdgesTourDiversity. Es un observador desarrollado para Scatter Search con el objetivo de obtener una información importante para el análisis del metaheurístico. Esta clase recopila las veces que se repite una misma artista en las soluciones que se generan en cada ciclo de la simulación. Los datos son volcado a un fichero para ser tratado por programas como Matlab.
- TimeSimCounter. Es un controlador que informa del tiempo real transcurrido en segundos en cada ciclo de la simulación y poder detener la

simulación en un momento determinado expresado en segundos también. La función principal de esta clase es conocer el tiempo real que ha consumido la ejecución de un experimento.

- Individual. Esta clase define a un camino, ciclo hamiltoniano o tour como un individuo u objeto y sobre el define algunas operaciones. Igualmente la existencia de esta clase se origina por el diseño orientado a objetos que sigue la implementación que aquí se ha realizado. Además de las operaciones que proporciona sobre los caminos, lo más interesante de esta clase es que nos permite declarar una serie de atributos sobre el objeto que pueden ser de gran ayuda en las implementaciones de los metaheurísticos; como es el caso de Scatter Search que hace uso de esta clase para poder marcar a los caminos como “viejos”, esto es, ya provienen de una generación anterior o marcarlos como “nuevos” si se han incorporado ahora en la generación actual.
- Edge. Esta clase participa en muchos heurísticos para dar soporte al concepto de arista en las diferentes implementaciones. Como este trabajo fin de grado se ha elaborado a partir de los principios del diseño orientado a objetos, era imprescindible definir esta clase para manejar a las aristas. Esta clase proporciona los constructores de la misma para crear las aristas, operaciones sobre ellas como devolver las ciudades que la componen, la longitud de la arista o intercambiar las ciudades en la arista. También se redefine el método toString para pasar la arista a una cadena de caracteres que pueden ser visualizados en pantalla. Quizás lo más importante de esta clase es que implementa la interfaz de JAVA comparator; de esta forma, estamos diciendo que la clase Edge tiene definido un orden y, en base, a ese orden las aristas pueden ser comparadas y ordenadas entre ellas como, por ejemplo, de números naturales se tratara. Por lo tanto, esta clase tiene que reescribir los métodos equals y compare que proporciona la interfaz comparator. Lo cual nos va a permitir el uso común a operadores como la igualdad (== ó equals) o los operadores de comparación mayor que (>) o menor que (<). O sus homólogos, mayor o igual que o menor o igual que. Es necesario advertir que el criterio de ordenación para la clase arista es la longitud o peso de la misma. Y la igualdad de dos arista se entiende si se componen de las dos mismas ciudades.

4.8.- CONFIGURACIÓN DE LAS SIMULACIONES

En este punto conocemos todas las clases desarrolladas y su cometido. El funcionamiento del simulador de redes P2P PeerSim, como hemos mencionado con anterioridad, es parecido a un framework donde la construcción de una simulación, como es el objetivo que queremos alcanzar aquí, trata básicamente de ensamblar de forma adecuada estas clases con la finalidad de ejecutar los algoritmos que formen parte del prototipo que vamos a simular.

Con este conocimiento en la mano, vamos a explicar como se ha realizado la configuración de los distintos prototipos que se han implementado para llevar a cabo los experimentos o pruebas con el simulador; siendo estos: Heurísticas básicas, Búsqueda Dispersa (*Scatter Search*) y Búsqueda Tabú (*Tabu Search*).

La disposición de las distintas clases que forman los prototipos se va a reflejar en este apartado de forma gráfica a través de los esquemas de red que se han visto en la figura 3.09 de este documento. Creemos este formato el más adecuado porque es mucho más didáctico y asimilable y se guarda fidelidad con la diversa literatura desarrollada por los autores de PeerSim; unificando, en cierta manera, la forma de adquirir el conocimiento relacionado con PeerSim.

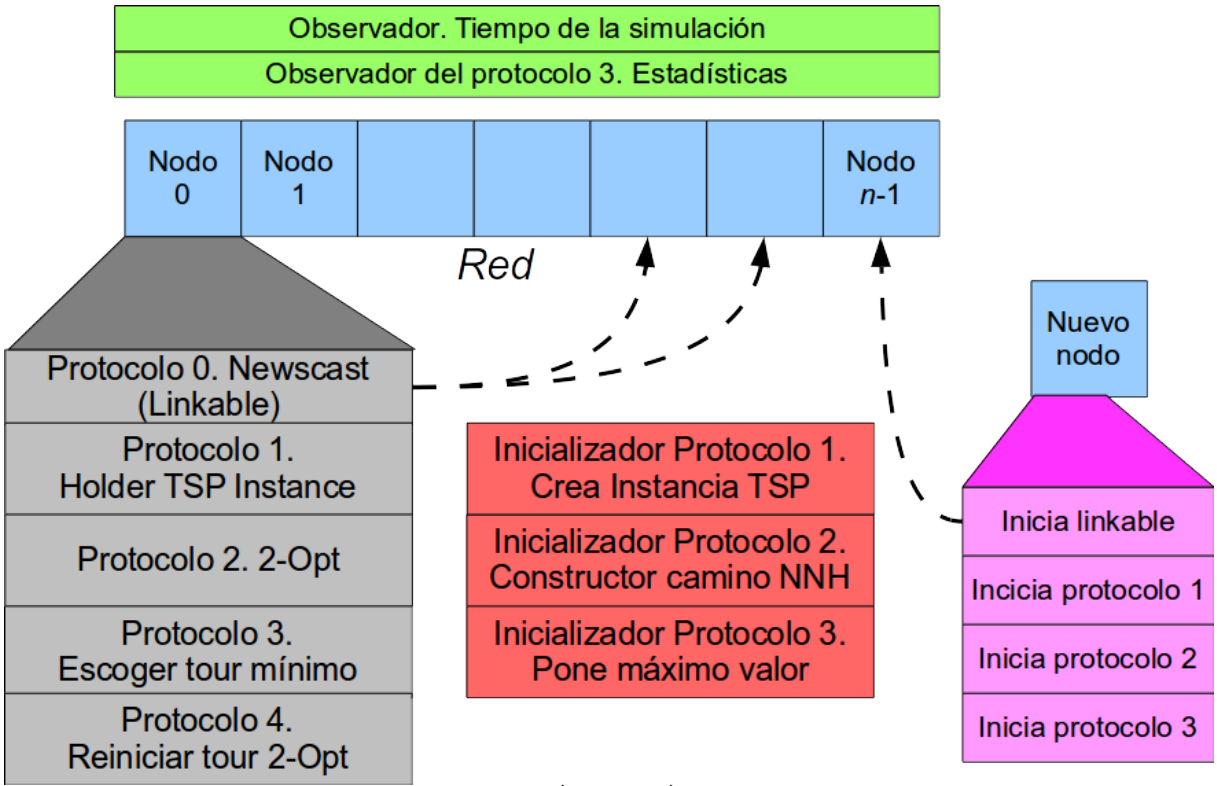


Fig 4.19 Esquema de la red de heurísticas básicas con constructor voraz (NNH)

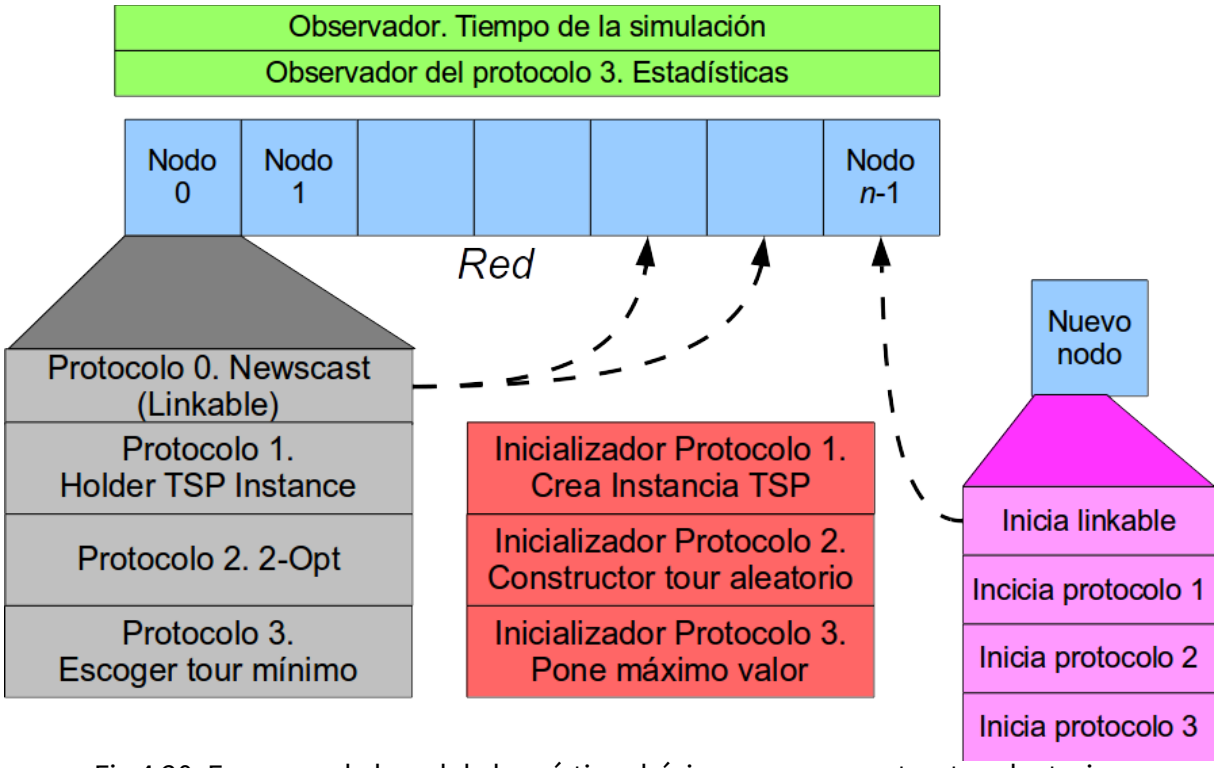


Fig 4.20. Esquema de la red de heurísticas básicas con un constructor aleatorios

Estas dos figuras corresponden a los esquemas de la red para los dos tipos de pruebas que se han realizado de heurísticas básicas. Un constructor para el camino inicial con el algoritmo de diseño ávido implementado de la heurística del vecino más cercano y el heurístico constructor de un camino aleatorio, respectivamente.

Estos experimentos se han llamado con este nombre de *heurísticas básicas* porque lo componen varios heurísticos desarrollados para el problema de viajante de comercio, los cuales son primera opción para el estudio de este problema y el desarrollo de otras estrategias en base a sus fundamentos.

A nivel de resolución del TSP con estos heurísticos, el planteamiento es muy sencillo: cargar la correspondiente instancia, generar un camino a través del heurístico de construcción (NNH o aleatorio) y mejorarlo con el algoritmo 2-Opt.

Desde el punto de vista de la simulación tenemos en color azul la representación de los nodos que componen la red. Todos ellos forman un vector o un array, constituyendo la red subyacente P2P. El motor de ciclos del simulador recorre por orden los componentes del array y ejecuta las clases necesarias. El tamaño del array es variable debido al dinamismo de la red.

En rojo se ha representado la torre de inicializadores que se ejecuta una

única vez al principio de la simulación en cada uno de los nodos de la red. Existe un inicializador por cada protocolo que compone la simulación. En nuestro caso, son muy importantes porque estos inicializadores van a cargar la instancia del TSP adecuada y van a construir el camino inicial.

Bajo el color gris se ilustra la pila de protocolos que compone cada nodo y que se ejecutará en cada ciclo. En ella, estará el protocolo que mantiene la topología de la red y los demás que realicen el cometido de la simulación, en nuestro caso, un protocolo que almacena una copia de la instancia del TSP cargada al inicio, un heurístico de mejora (2-Opt) y el protocolo que comunica y pone disponible el recurso a compartir (el camino o tour).

Los controladores y los observadores que afectan a toda la red se han representado en color verde. La inclusión de un nuevo nodo debido al dinamismo de la red, requiere de una inicialización previa antes de incorporarse a la red, estos inicializadores se encuentran en color magenta.

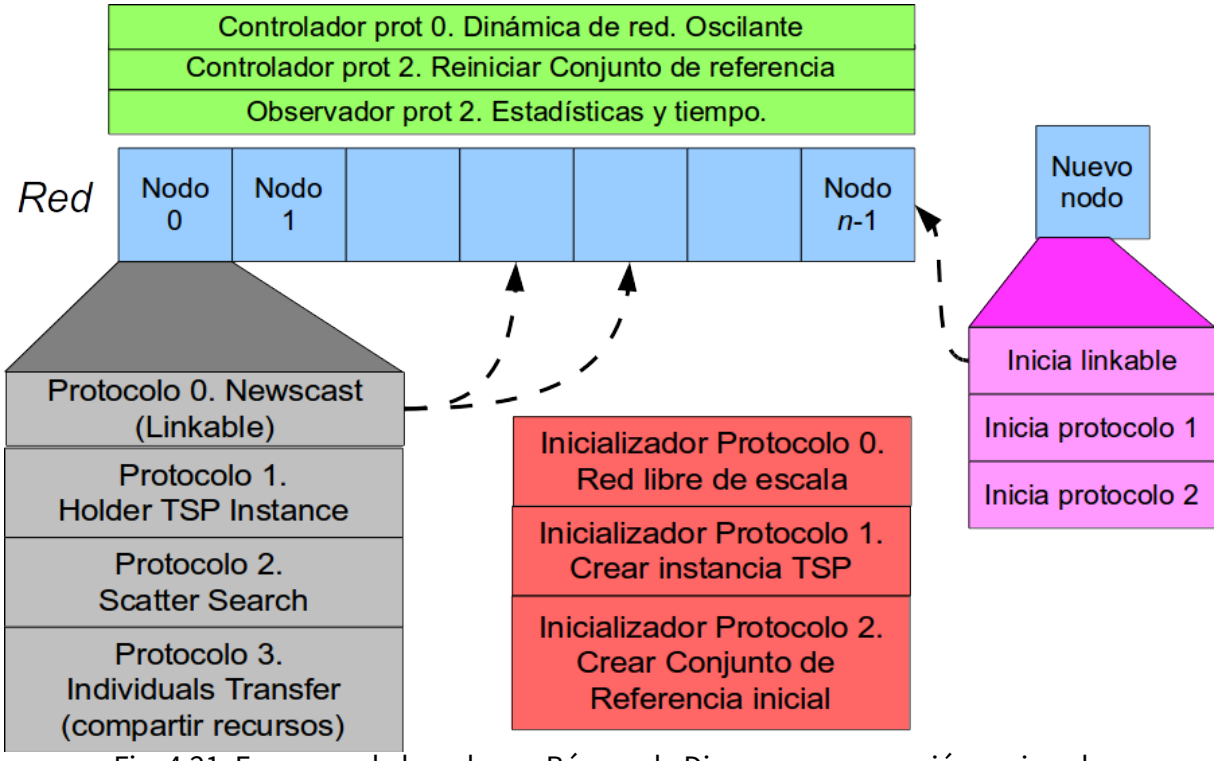


Fig. 4.21. Esquema de la red para Búsqueda Dispersa en su versión mejorada

Con las diferencias propias de Scatter Search este diagrama sigue la misma metodología que los anteriores. A destacar, el observador que reinicia el conjunto de referencia, con el cual podremos conmutar entre la versión estándar y mejorada del algoritmo, si no se encuentra presente en la simulación o sí lo

está, respectivamente.

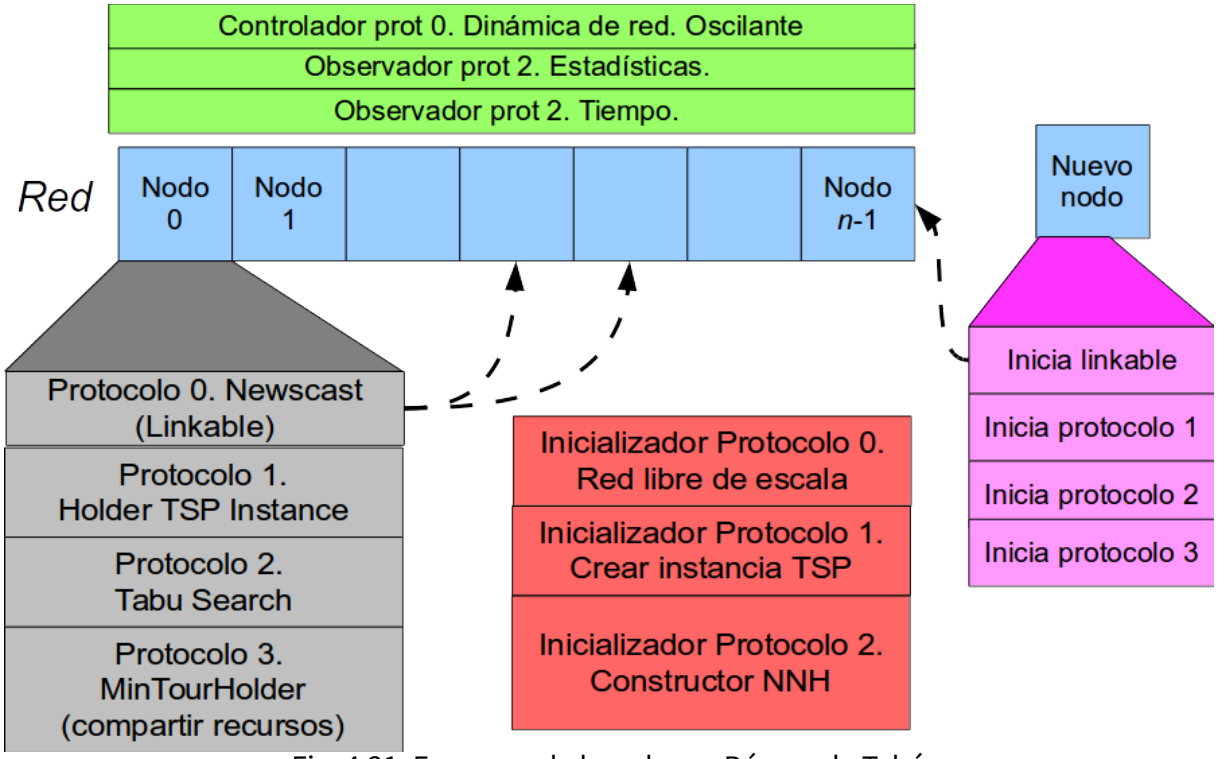


Fig. 4.21. Esquema de la red para Búsqueda Tabú

Como Búsqueda Tabú es un metaheurístico de mejora para el TSP, este esquema es prácticamente igual al de heurísticas básicas.

CAPÍTULO 5. ANÁLISIS DE LOS METAHEURÍSTICOS EN REDES P2P.

5.1.- EL REPOSITORIO TSPLIB.

El repositorio universal TSPLIB es una recopilación de instancias de ejemplo para el problema del viajante de comercio procedente de fuentes diferentes y albergando instancias de varios tipos de problemas para el TSP, según así lo define su principal mantenedor Gerhad Reinelt de la Universidad de Heidelberg. Además, aloja a otros tipos de problemas que presentan alguna similitud con el TSP, como el problema de la ruta de vehículos y el problema de la ordenación secuencial.

Este repositorio es referencia universal para todos los investigadores que se dedican al estudio del TSP, ya que, contiene instancias que fácilmente pueden ser compartidas con otros al ser de acceso público y, además, un gran número de ellas se encuentran resueltas hasta el óptimo. Otras, sin embargo, tienen el mejor resultado obtenido hasta el momento.

En este trabajo se ha hecho uso, para el desarrollo y evaluación de los heurísticos implementados, de un subconjunto de instancias contenidas en la TSPLIB. En concreto, se han tomado de este repositorio las instancias que se encuentran clasificadas para el TSP simétrico y estén definidas dentro de un espacio euclidiano bidimensional. Los ficheros que contienen este tipo de instancias, en general todas, tienen un formato especial que las clases implementadas aquí para el manejo de instancias interpretan correctamente, como por ejemplo el siguiente:

```
NAME : d198      // NOMBRE DE LA INSTANCIA
COMMENT : Drilling problem (Reinelt) //PROCEDENCIA-AUTOR
TYPE : TSP      // PROBLEMA PERTENECIENTE AL TSP
DIMENSION: 198  // NÚMERO DE CIUDADES DE LA INSTANCIA.
EDGE_WEIGHT_TYPE : EUC_2D // TIPO DE TSP: EUCLIDIANO Y 2D
NODE_COORD_SECTION  // ANUNCIA COMIENZO DE LAS COORDENADAS
```

```
1 0.00000e+00 0.00000e+00  // COORDENADAS CIUDAD 1 (X,Y)
2 5.51200e+02 9.96400e+02  // CIUDAD 2 (5.512E2,9.964E2)
3 6.27400e+02 9.96400e+02
...
197 4.02830e+03 1.01030e+03
198 3.95210e+03 1.01030e+03 // ÚLTIMA CIUDAD
EOF  // FLAG DE FIN DE FICHERO
```

A continuación vemos el listado de instancias que se han utilizado para este proyecto.

berlin52	lin318
lin105	fl417
ch150	d493
rat195	u574
d198	p654
kroB200	pr1002
gil262	pcb1173
pr264	u2152
a280	

El nombre de las instancias suele estar compuesto por unas letras que hacen referencia al origen del problema desde el que proceden y un número que advierte del tamaño de la instancia.

5.2.- METODOLOGÍA.

La evaluación de los experimentos simulados en PeerSim se ha realizado sobre una máquina con un procesador de la familia x86-64 de doble núcleo cuya marca y modelo son: Intel Core 2 Duo a una frecuencia de reloj de 1666 Mhz y 2 Gigabytes de memoria RAM. El sistema operativo de esta máquina ha sido GNU/Linux en su distribución Ubuntu, versión 14.04 de 64 bits.

La codificación se ha desarrollado bajo el IDE Eclipse en la versión Indigo 3.7.2. La versión de la máquina de JAVA utilizado es 1.7. El código implementado

es compatible con esta misma versión de JAVA, es decir, la 1.7 y también con las anteriores 1.6 y 1.5. Con versiones más antiguas de la máquina de JAVA puede haber problemas en algunas implementaciones, ya que, se hacen uso de algunas características que sólo están presentes en la versiones anteriormente citadas, como por ejemplo, el uso de tipos genéricos.

El simulador de redes P2P PeerSim utilizado está en su última versión 1.0.5. Según sus autores el desarrollo de PeerSim no va a continuar en un futuro próximo, pero debido a su carácter abierto, numerosos investigadores siguen aportando contribuciones para ampliar el número de protocolos disponibles, así como controladores para el modelado de la red. En la siguiente página del proyecto existe una relación de estas contribuciones.

http://peersim.sourceforge.net/#code

La evaluación de cada experimento ha consistido en ejecutar en el simulador el código implementado para la batería de instancias seleccionadas desde la TSPLIB y que se encuentra listada en el apartado anterior. En ella se ha tomado, por norma general, la desviación típica respecto al óptimo, puesto que, éste es conocido y se encuentra descrito en el repositorio TSPLIB y el tiempo de ejecución en segundos. La desviación se ha calculado con la siguiente fórmula, ofreciendo un porcentaje:

$$desv_{heurístico} = \frac{valor_{heurístico} - valor_{óptimo}}{valor_{óptimo}} \cdot 100$$

Fig. 5.1. Fórmula para la obtención de la desviación respecto al óptimo.

5.3.- RESULTADOS

En este apartado se van a mostrar los resultados que se han obtenido de evaluar cada uno de los experimentos que componen el objetivo de este trabajo fin de grado. En general, los resultados que se publicarán constarán de una tabla que indicará la desviación típica respecto al valor óptimo que se calcula por la fórmula expuesta en el apartado anterior para cada instancia y el tiempo medio de ejecución del heurístico para la instancia correspondiente. Además se mostrará la configuración inicial realizada para cada simulación y las veces que se ha repetido el mismo para obtener una población de muestras importantes de la instancia y que reflejen un resultado representativo.

5.3.1.- Heurísticas básicas con constructor aleatorio

Estos experimentos conjugan un juego de heurísticas cuya formulación es muy sencilla; aunque no por ello tengan una mala eficiencia. Por ello se le ha llamado de heurísticas básicas. Los experimentos se han realizado en dos grandes grupos. Un constructor de caminos aleatorio, caso de este apartado, y otro empleando una heurística de construcción muy popular como es 2-Opt, para el siguiente apartado.

En cada apartado se han realizado dos experimentos por cada una de las instancias seleccionadas para este trabajo (indicadas en el apartado 5.1), donde el primero la simulación parará cuando se repita el mismo camino mínimo $10m$ veces, siendo m el tamaño de la red. El segundo experimento culminará cuando se repita el camino mínimo n^2,siendo n el número de ciudades.

Todos estos experimentos se han repetido con estos mismos parámetros con un tamaño de red inicial de 32, 64, 128 y 256 ciudades.

Veamos a continuación el contenido de un fichero de configuración para el simulador PeerSim para un experimento de este apartado. En este caso particular los parámetros de la red son los siguientes:

- Instancia obtenida del repositorio TSPLIB “berlin52”
- Un tamaño de red de 32 ciudades iniciales
- Una dinámica de red probabilística para $1/10m$, siendo m el tamaño de la red.
- Parada de la simulación: repetición del mismo camino mínimo $10m$ veces, siendo m el tamaño de la red.

```
CYCLES 2704
LENGTH 52
NETSIZE 32
CACHE 30
K 28

random.seed 123456789
simulation.cycles CYCLES
```

```
network.size NETSIZE
network.node peersim.core.GeneralNode

# Initializers ----------------

# Linkable initializer
init.freeba WireScaleFreeBA
init.freeba.protocol lnk
init.freeba.k K

init.etsp FileETSP
init.etsp.file /home/paco/berlin52.tsp
init.etsp.protocol problem

init.constructor RandomConstructor
init.constructor.instance problem
init.constructor.size LENGTH
init.constructor.protocol improve

init.mininit MinTourInitializer
init.mininit.size LENGTH
init.mininit.protocol min
init.mininit.source improve

include.init freeba etsp constructor mininit

# Protocols ------------------------------

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache CACHE

protocol.problem TSPHolder

protocol.improve TwoOpt
protocol.improve.instance problem
protocol.improve.size LENGTH

protocol.min MinTourHolder
protocol.min.linkable lnk
protocol.min.times 10
protocol.min.protocol improve
protocol.min.instance problem

protocol.restart MinTourRestart
protocol.restart.instance problem
protocol.restart.protocol min
protocol.restart.improve improve
protocol.restart.max 50
protocol.restart.min 20
```

```
# Controller and Observers -------------------------

control.mino MinTourObserver
control.mino.protocol min
control.mino.file true
control.mino.filename out-rand-2opt-berlin52-32-110n-10n.txt

control.minv MinTourValueObserver
control.minv.protocol min
control.minv.multiple 10

control.dnet ProbabilityNetwork
control.dnet.maxsize NETSIZE
control.dnet.multiple 10
control.dnet.init.0 ScaleFreeNI
control.dnet.init.0.protocol lnk
control.dnet.init.0.k K
control.dnet.init.1 TSPHolderNI
control.dnet.init.1.protocol problem
control.dnet.init.2 RandomImproversNI
control.dnet.init.2.protocol improve
control.dnet.init.3 MinTourNI
control.dnet.init.3.protocol min
control.dnet.init.3.improve improve
```

Debido al tiempo para procesar estas simulaciones, los experimentos se han repetido 10 veces de la siguiente manera:

- Cuando la condición de parada es la repetición del mismo resultado, se han repetido 10 veces desde la instancia *berlin52* hasta *d493*. Para el resto, las de mayor tamaño, existe un único experimento.
- Cuando la condición de parada es igualar el número de ciclos al tamaño de la instancia al cuadrado, donde el tiempo de simulación es muy superior al anterior criterio de parada, se han repetido los experimentos 10 veces desde *berlin52* hasta *a280*. El resto un solo experimento.

Esto principalmente ha repercutido en el cálculo del error estándar donde el cociente cambia entre el número de experimento en el caso de que exista repeticiones o el número de muestras (tamaño de la red) en el caso de un único experimento.

A continuación se presentarán una serie de tablas con la información proporcionada por los observadores:

Inicialización: Aleatoria Tamaño de la red: 32 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	93,54	29,58	8750,10	00:00:03	0,00	1364
lin105	14379	14412	14695,07	14681	14878	211,70	66,95	44816,93	00:00:05	0,23	872
ch150	6528	6684	6935,69	6854	8874	127,87	40,44	16351,08	00:00:09	2,39	737
rat195	2323	2448	2526,21	2523	2616	45,41	14,36	2062,31	00:00:07	5,38	375
d198	15780	16089	16267,00	16267	16267	2187,73	691,82	###	00:00:18	1,96	936
kroB200	29437	30263	31262,20	31234	32080	358,01	113,21	128172,00	00:00:13	2,81	646
gil262	2378	2506	2607,00	2607	2607	163,50	51,70	26733,14	00:00:12	5,38	363
pr264	49135	50388	51001,00	51001	51001	416,86	131,82	173775,15	00:00:26	2,55	747
a280	2579	2708	2755,00	2755	2755	116,50	36,84	13571,98	00:00:37	5,00	929
lin318	42029	44122	44675,00	44675	44675	341,48	107,98	116606,59	00:00:39	4,98	734
fl417	11861	12055	12213,00	12213	12213	2161,52	683,53	###	00:01:21	1,64	857
d493	35002	37098	37469,00	37469	37469	365,52	115,59	133604,10	00:02:31	5,99	1086
		Resultados para un experimento									
u574	36902	40236	40464,82	40236	46643	1210,81	214,04	###	00:02:38	9,03	825
p654	34643	35281	35281,00	35281	35281	0,00	0,00	0,00	00:02:57	1,84	678
pr1002	259045	284263	284263,00	284263	284263	0,00	0,00	0,00	00:04:03	9,73	375
pcb1173	56892	63071	63071,00	63071	63071	0,00	0,00	0,00	00:12:02	10,86	789
u2152	64253	72874	72936,11	72874	74613	328,64	58,10	108004,32	00:30:53	13,42	584

Inicialización: Aleatoria Tamaño de la red: 32 Volatilidad de la red: 1/10n Tiempo sim: 10*n											
INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7618,00	7618	7618	348,94	110,34	121757,16	00:00:01	0,00	758
lin105	14379	14536	14590,00	14590	14590	0,00	0,00	0,00	00:00:06	1,09	909
ch150	6528	6717	6749,00	6749	6749	59,35	18,77	3522,57	00:00:07	2,90	607
rat195	2323	2435	3133,58	2498	22201	355,58	112,45	126438,97	00:00:25	4,82	1233
d198	15780	16081	16233,00	16233	16233	0,00	0,00	0,00	00:00:06	1,91	331
kroB200	29437	30378	30858,00	30858	30858	59,98	18,97	3597,08	00:00:18	3,20	864
gil262	2378	2493	2568,00	2568	2568	443,23	140,16	196454,79	00:00:20	4,84	553
pr264	49135	50057	50691,71	50565	54493	37654,50	11907,40	1417861597,48	00:00:32	1,88	880
a280	2579	2683	2770,00	2770	2770	11,07	3,50	122,44	00:00:32	4,03	778
lin318	42029	44056	44657,35	44536	48298	10289,46	3253,81	105873059,24	00:00:33	4,82	607
fl417	11861	12082	12132,00	12132	12132	9,36	2,96	87,56	00:01:21	1,86	827
d493	35002	37017	37584,00	37584	37584	34,50	10,91	1190,40	00:01:05	5,76	443
		Resultados de un experimento									
u574	36902	40306	59768,41	40306	663103	110096,00	19462,41	12121128225,28	00:02:36	9,22	770
p654	34643	35392	35392,00	35392	35392	0,00	0,00	0,00	00:04:08	2,16	918
pr1002	259045	283259	283259,00	283259	283259	0,00	0,00	0,00	00:11:43	9,35	1037
pcb1173	56892	63605	63605,00	63605	63605	0,00	0,00	0,00	00:11:07	11,80	696
u2152	64253	73581	153427,94	73581	2548836	444568,92	78589,42	197641526291,05	00:21:50	14,52	399

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 32 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	152,32	48,17	23202,55	00:00:05	0,00	2704
lin105	14379	14379	14468,00	14468	14468	1352,54	427,71	1829366,36	00:01:07	0,00	11025
ch150	6528	6617	6665,00	6665	6665	114,51	36,21	13113,05	00:04:30	1,36	22500
rat195	2323	2399	2431,00	2431	2431	39,59	12,52	1567,36	00:12:21	3,27	38025
d198	15780	15959	16361,63	16039	17117	1114,05	352,29	1241104,94	00:13:03	1,13	39204
kroB200	29437	29908	30269,00	30269	30269	2152,70	680,74	4634122,59	00:13:32	1,60	40000
gil262	2378	2466	2538,37	2498	2671	70,92	22,43	5029,12	00:45:17	3,70	68644
pr264	49135	49475	49776,00	49776	49776	1296,49	409,99	1680884,94	00:40:58	0,69	69696
a280	2579	2657	2738,45	2660	3092	117,47	37,15	13798,98	00:51:53	3,02	78400
		Resultados para un experimento									
lin318	42029	43616	43798,61	43616	47571	764,17	135,09	583956,31	01:27:57	3,78	101124
fl417	11861	12028	12397,23	12028	13035	493,56	87,25	243604,87	04:33:15	1,41	173889
d493	35002	36750	37528,16	36750	38943	1066,62	188,55	1137671,80	09:29:43	4,99	243049
u574	36902	39384	40131,29	39384	41166	893,87	158,02	799002,81	17:35:59	6,73	329476

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 32 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	0,00	0,00	0,00	00:00:06	0,00	2704
lin105	14379	14379	14434,00	14434	14434	62,77	19,85	3940,65	00:01:12	0,00	11025
ch150	6528	6628	6709,88	6665	7275	110,83	35,05	12283,36	00:04:43	1,53	22500
rat195	2323	2415	2441,22	2430	2645	24,40	7,72	595,50	00:12:57	3,96	38025
d198	15780	15933	16043,00	16043	16043	178,16	56,34	31741,81	00:13:30	0,97	39204
kroB200	29437	30112	30479,68	30244	36112	1275,76	403,43	1627564,19	00:14:25	2,29	40000
gil262	2378	2462	2478,00	2478	2478	39,29	12,42	1543,46	00:43:38	3,53	68644
pr264	49135	49410	49807,69	49646	54820	227,93	72,08	51952,75	00:43:46	0,56	69696
a280	2579	2648	2661,19	2648	2859	31,15	9,85	970,19	00:54:36	2,68	78400
		Resultados para un experimento									
lin318	42029	43517	43615,38	43517	46665	556,49	98,37	309684,50	01:32:19	3,54	101124
fl417	11861	12014	12014,00	12014	12014	0,00	0,00	0,00	04:56:47	1,29	173889
d493	35002	36515	36515,00	36515	36515	0,00	0,00	0,00	09:55:42	4,32	243049
u574	36902	39416	39482,34	39416	41539	375,30	66,34	140847,78	19:05:39	6,81	329476

Inicialización: Aleatoria Tamaño de la red: 64 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	60,62	19,17	3674,20	00:00:04	0,00	1057
lin105	14379	14379	14510,89	14493	15584	595,78	188,40	354948,67	00:00:31	0,00	2564
ch150	6528	6694	6761,26	6753	7096	211,57	66,90	44762,07	00:00:32	2,54	1339
rat195	2323	2433	2461,00	2461	2461	102,34	32,36	10472,99	00:01:08	4,74	1719
d198	15780	16080	16181,03	16170	16854	164,08	51,89	26923,51	00:00:58	1,90	1424
kroB200	29437	30289	30352,00	30352	30352	1215,59	384,40	1477657,23	00:01:03	2,89	1524
gil262	2378	2495	2523,51	2522	2614	124,11	39,25	15403,70	00:01:28	4,92	1234
pr264	49135	49549	49606,66	49549	51250	8964,85	2834,93	80368538,99	00:01:52	0,84	1553
a280	2579	2690	2756,25	2738	3109	172,87	54,66	29882,31	00:01:58	4,30	1448
lin318	42029	44019	44421,00	44421	44421	2945,93	931,59	8678510,25	00:01:53	4,73	1063
fl417	11861	12097	12131,79	12123	12391	1437,05	454,44	2065124,11	00:04:35	1,99	1434
d493	35002	36984	37410,05	37201	39086	2683,68	848,65	7202144,39	00:06:27	5,66	1370
		Resultados para un experimento									
u574	36902	40090	40112,93	40090	41443	176,15	22,02	31027,27	00:09:30	8,64	1453
p654	34643	35266	35322,35	35266	38647	436,49	54,56	190519,35	00:13:28	1,80	1522
pr1002	259045	282318	282658,13	282318	292522	1847,13	230,89	3411894,76	00:32:44	8,98	1471
pcb1173	56892	63325	63325,00	63325	63325	0,00	0,00	0,00	00:41:18	11,31	1322
u2152	64253	73167	73197,97	73167	75056	241,86	30,23	58497,07	01:54:50	13,87	1060

Inicialización: Aleatoria Tamaño de la red: 64 Volatilidad de la red: 1/10n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	1485,16	469,65	2205692,78	00:00:04	0,00	1099
lin105	14379	14379	14482,00	14482	14482	14,93	4,72	222,89	00:00:34	0,00	1687
ch150	6528	6657	7519,50	6752	54337	1773,31	560,77	3144632,69	00:00:42	1,98	1640
rat195	2323	2440	2444,00	2444	2444	746,60	236,09	557407,37	00:00:47	5,04	1154
d198	15780	16070	16151,00	16151	16151	4389,37	1388,04	19266556,08	00:00:57	1,84	1365
kroB200	29437	30281	30446,00	30446	30446	24,94	7,89	621,88	00:01:40	2,87	2338
gil262	2378	2507	2899,95	2530	26207	1861,29	588,59	3464390,22	00:01:21	5,42	1104
pr264	49135	49437	50637,00	50637	50637	13089,92	4139,40	171345942,92	00:01:49	0,61	1469
a280	2579	2682	2712,00	2712	2712	781,14	247,02	610178,41	00:02:44	3,99	1966
lin318	42029	43558	44141,00	44141	44141	8837,59	2794,69	78102950,93	00:03:31	3,64	1933
fl417	11861	12063	19704,52	12095	499104	6093,95	1927,08	37136223,00	00:03:53	1,70	1173
d493	35002	36842	36842,00	36842	36842	5265,30	1665,03	27723353,40	00:10:06	5,26	2068
		Resultados para un experimento									
u574	36902	40219	40219,00	40219	40219	0,00	0,00	0,00	00:08:13	8,99	1157
p654	34643	35313	66224,27	35313	1982723	245350,60	30668,82	60196915997,84	00:11:07	1,93	1181
pr1002	259045	283848	283848,00	283848	283848	0,00	0,00	0,00	00:57:59	9,57	2563
pcb1173	56892	63055	63055,00	63055	63055	0,00	0,00	0,00	00:45:22	10,83	1415
u2152	64253	72059	72059,00	72059	72059	0,00	0,00	0,00	03:49:09	12,15	2057

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 64 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7556,85	7542	8463	81,36	25,73	6620,21	00:00:09	0,00	2704
lin105	14379	14379	14434,05	14416	15535	341,02	107,84	116294,87	00:02:23	0,00	11025
ch150	6528	6595	6667,83	6656	7401	308,93	97,69	95438,72	00:09:05	1,03	22500
rat195	2323	2406	2428,02	2408	2591	144,95	45,84	21009,63	00:25:13	3,57	38025
d198	15780	15958	16093,03	16030	16811	1497,14	473,44	2241433,59	00:26:23	1,13	39204
kroB200	29437	29965	29965,00	29965	29965	3069,21	970,57	9420077,59	00:27:43	1,79	40000
gil262	2378	2468	2606,57	2472	8857	147,36	46,60	21715,31	01:22:52	3,78	68644
pr264	49135	49419	50262,89	49858	55836	7276,64	2301,08	52949499,47	01:33:56	0,58	69696
a280	2579	2642	2669,83	2645	2872	138,38	43,76	19149,23	01:46:30	2,44	78400
		Resultados para un experimento									
lin318	42029	43704	43972,78	43704	48327	834,83	104,35	696946,98	03:00:39	3,99	101124
fl417	11861	12041	12156,22	12041	16556	590,14	73,77	348269,08	09:22:18	1,52	173889
d493	35002	36515	36718,71	36515	38620	627,42	78,43	393660,60	19:48:30	4,32	243049
u574	36902	38948	39177,94	38948	41666	720,69	90,09	519391,06	35:56:53	5,54	329476

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 64 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 expermientos									
berlin52	7542	7542	7542,00	7542	7542	0,00	0,00	0,00	00:01:04	0,00	2704
lin105	14379	14379	14508,60	14379	16012	315,73	99,84	99683,17	00:02:27	0,00	11025
ch150	6528	6584	6633,00	6633	6633	26,55	8,40	704,79	00:09:01	0,86	22500
rat195	2323	2379	2422,94	2420	2608	17,13	5,42	293,32	00:25:53	2,41	38025
d198	15780	15959	16058,00	16058	16058	66,56	21,05	4430,57	00:27:07	1,13	39204
kroB200	29437	30006	30127,00	30127	30127	187,65	59,34	35212,52	00:28:32	1,93	40000
gil262	2378	2456	2497,00	2497	2497	9,93	3,14	98,67	01:24:29	3,28	68644
pr264	49135	49302	49409,00	49409	49409	360,30	113,94	129818,06	01:26:12	0,34	69696
a280	2579	2639	2668,00	2668	2668	10,55	3,34	111,33	01:49:14	2,33	78400
		Resultados para un experimento									
lin318	42029	43471	43576,34	43471	47002	591,85	73,98	350289,75	03:04:51	3,43	101124
fl417	11861	12013	12013,00	12013	12013	0,00	0,00	0,00	09:38:48	1,28	173889
d493	35002	36467	36486,36	36467	37706	154,88	19,36	23986,27	20:09:45	4,19	243049
u574	36902	39369	39443,75	39369	41761	419,48	52,44	175963,87	37:12:49	6,69	329476

Inicialización: Aleatoria Tamaño de la red: 128 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7550,57	7542	8199	130,67	41,32	17075,92	00:00:13	0,00	1630
lin105	14379	14401	14438,75	14426	15353	141,31	44,69	19968,17	00:01:11	0,15	1906
ch150	6528	6610	6660,04	6654	7403	92,20	29,16	8500,40	00:02:38	1,26	3260
rat195	2323	2430	2463,99	2456	2631	109,53	34,64	11996,58	00:02:44	4,61	2033
d198	15780	15966	16049,10	16033	16989	473,31	149,67	224020,29	00:04:08	1,18	3035
kroB200	29437	30131	30733,09	30622	33560	1007,95	318,74	1015953,97	00:03:28	2,36	2472
gil262	2378	2486	2494,69	2491	2693	95,66	30,25	9150,59	00:06:34	4,54	2707
pr264	49135	49573	49629,98	49597	53719	920,07	290,95	846532,88	00:04:49	0,89	1974
a280	2579	2653	2687,63	2681	2898	86,41	27,33	7467,13	00:07:09	2,87	2599
lin318	42029	43364	44298,69	44134	48531	2745,66	868,25	7538631,05	00:10:04	3,18	2791
fl417	11861	12027	12157,73	12065	17501	1204,64	380,94	1451164,13	00:20:56	1,40	3191
d493	35002	36715	36999,86	36928	38893	1643,58	519,75	2701368,11	00:23:37	4,89	2473
		Resultados para 1 experimento									
u574	36902	39836	39863,87	39836	42764	266,80	23,58	71184,11	00:57:41	7,95	4357
p654	34643	35121	35195,20	35121	39083	513,45	45,38	263627,76	00:47:44	1,38	2661
pr1002	259045	282435	282983,37	282435	294587	2462,39	217,65	6063370,20	01:34:31	9,03	2106
pcb1173	56892	62748	62765,79	62748	64972	198,92	17,58	39569,41	02:29:39	10,29	2374
u2152	64253	72335	72354,40	72335	74760	216,90	19,17	47045,00	07:14:58	12,58	1964

Inicialización: Aleatoria Tamaño de la red: 128 Volatilidad de la red: 1/10n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	900,79	284,86	811425,93	00:00:12	0,00	1817
lin105	14379	14379	14438,00	14438	14438	2645,71	836,65	6999783,67	00:01:17	0,00	2744
ch150	6528	6612	7050,80	6678	54396	2124,41	671,80	4513131,57	00:01:23	1,29	1723
rat195	2323	2425	2467,00	2467	2467	430,06	136,00	184951,67	00:04:47	4,39	3517
d198	15780	15921	16046,00	16046	16046	4644,89	1468,84	21574979,50	00:04:58	0,89	3581
kroB200	29437	30089	30448,00	30448	30448	5274,66	1667,99	27822070,12	00:03:36	2,21	2540
gil262	2378	2483	2521,00	2521	2521	301,00	95,19	90602,58	00:12:20	4,42	5025
pr264	49135	49620	49909,00	49909	49909	12675,12	4008,22	160658675,37	00:12:04	0,99	4888
a280	2579	2643	2701,00	2701	2701	1137,56	359,73	1294037,57	00:07:26	2,48	2673
lin318	42029	43585	44137,00	44137	44137	14752,62	4665,19	217639813,81	00:12:16	3,70	3360
fl417	11861	12037	12099,00	12099	12099	5290,58	1673,03	27990245,82	00:14:34	1,48	2206
d493	35002	36594	37118,00	37118	37118	12270,95	3880,41	150576132,83	00:26:29	4,55	2722
		Resutados para un experimento									
u574	36902	39827	39827,00	39827	39827	0,00	0,00	0,00	00:41:44	7,93	3093
p654	34643	35211	35211,00	35211	35211	0,00	0,00	0,00	00:37:03	1,64	2028
pr1002	259045	281949	281949,00	281949	281949	0,00	0,00	0,00	01:14:20	8,84	1633
pcb1173	56892	63139	63139,00	63139	63139	0,00	0,00	0,00	03:52:19	10,98	3613
u2152	64253	72169	72169,00	72169	72169	0,00	0,00	0,00	16:56:24	12,32	4520

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 128 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7549,97	7542	7797	63,34	20,03	4011,70	00:00:17	0,00	2704
lin105	14379	14379	14422,98	14379	16468	356,91	112,86	127382,31	00:04:33	0,00	11025
ch150	6528	6581	6735,82	6626	19619	359,65	113,73	129349,45	00:17:52	0,81	22500
rat195	2323	2404	2463,51	2404	8605	129,41	40,92	16746,76	00:51:07	3,49	38025
d198	15780	15905	16066,09	16017	17540	1153,20	364,67	1329863,50	00:53:54	0,79	39204
kroB200	29437	29689	29844,46	29689	32118	2237,27	707,49	5005390,26	00:56:00	0,86	40000
gil262	2378	2456	2468,75	2463	2647	103,14	32,62	10638,00	02:46:32	3,28	68644
pr264	49135	49319	49998,59	49429	67642	1145,56	362,26	1312316,79	02:52:15	0,37	69696
a280	2579	2648	2653,24	2648	3032	220,95	69,87	48817,95	03:35:58	2,68	78400
		Resultados para un experimento									
lin318	42029	43353	43544,40	43353	46378	659,08	82,38	434382,24	06:05:03	3,15	101124
fl417	11861	12007	12194,97	12007	18549	683,28	85,41	466865,15	18:49:56	1,23	173889
d493	35002	36501	36614,08	36501	39629	569,97	71,25	324860,42	39:19:00	4,28	243049
u574	36902	39288	39367,24	39288	41856	346,06	43,26	119757,49	73:04:46	6,47	329476

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 128 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	420,68	133,03	176974,32	00:00:18	0,00	2704
lin105	14379	14379	14379,00	14379	14379	31,06	9,82	964,70	00:04:36	0,00	11025
ch150	6528	6554	6573,00	6573	6573	52,47	16,59	2752,82	00:18:07	0,40	22500
rat195	2323	2402	2413,00	2413	2413	13,36	4,23	178,60	00:51:58	3,40	38025
d198	15780	15961	15995,00	15995	15995	49,72	15,72	2471,83	00:54:12	1,15	39204
kroB200	29437	29900	30045,24	30014	34013	174,55	55,20	30468,23	00:56:48	1,57	40000
gil262	2378	2460	2473,38	2472	2560	5,89	1,86	34,73	02:49:04	3,45	68644
pr264	49135	49215	49553,70	49469	54890	277,97	87,90	77264,83	02:51:57	0,16	69696
a280	2579	2634	2639,00	2639	2639	5,26	1,66	27,67	03:39:04	2,13	78400
		Resultados para un experimentos									
lin318	42029	43432	43513,77	43432	47012	529,96	45,11	280859,36	06:10:13	3,34	101124
fl417	11861	12018	12022,98	12018	12655	56,30	4,79	3170,07	19:06:59	1,32	173889
d493	35002	36518	36518,00	36518	36518	0,00	0,00	0,00	39:44:27	4,33	243049
u574	36902	39213	39247,95	39213	41450	278,52	23,71	77574,47	84:42:31	6,26	329476

Inicialización: Aleatoria Tamaño de la red: 256 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7572,40	7542	12231	115,23	36,44	13277,44	00:00:33	0,00	2669
lin105	14379	14379	14465,13	14412	20632	208,01	65,78	43268,43	00:02:38	0,00	2736
ch150	6528	6605	6689,53	6643	15991	233,01	73,69	54295,63	00:08:53	1,18	5598
rat195	2323	2409	2429,14	2429	2464	27,79	8,79	772,48	00:12:04	3,70	4437
d198	15780	15887	16018,28	16013	16770	384,34	121,54	147718,26	00:18:39	0,68	6784
kroB200	29437	29945	30022,04	30009	33320	667,66	211,13	445775,20	00:10:16	1,73	3632
gil262	2378	2452	2522,51	2491	9807	69,32	21,92	4804,81	00:38:28	3,11	7880
pr264	49135	49251	49537,26	49479	55899	2778,50	878,64	7720086,15	00:31:38	0,24	6452
a280	2579	2643	2662,00	2662	2662	62,82	19,87	3946,96	00:22:23	2,48	4036
lin318	42029	43455	43614,86	43591	47154	496,82	157,11	246825,98	00:48:58	3,39	6711
fl417	11861	11978	12027,83	12016	14010	2126,80	672,55	4523272,20	00:50:02	0,99	3811
d493	35002	36440	36741,06	36721	38858	1560,68	493,53	2435723,49	01:03:58	4,11	3313
		Resultados para un experimento									
u574	36902	39534	39573,79	39534	41880	286,00	17,88	81797,72	02:39:11	7,13	5950
p654	34643	35113	35121,83	35113	37347	140,45	8,78	19726,31	03:58:34	1,36	6586
pr1002	259045	280754	281303,76	280754	419844	8744,51	546,53	76466514,23	06:53:41	8,38	4599
pcb1173	56892	62240	62252,70	62240	65441	201,64	12,60	40660,32	20:21:10	9,40	9501
u2152	64253	72440	72459,75	72440	75190	222,83	13,93	49654,02	01:54:49	12,74	3479

Inicialización: Aleatoria Tamaño de la red: 256 Volatilidad de la red: 1/10n Tiempo sim: 10*n											
INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7543,30	7542	7875	31,04	9,82	963,71	00:00:34	0,00	2703
lin105	14379	14379	14401,00	14401	14401	663,21	209,73	439846,31	00:03:45	0,00	4656
ch150	6528	6586	6652,09	6649	7436	10,60	3,35	112,27	00:10:59	0,89	6893
rat195	2323	2406	2432,64	2432	2595	1,02	0,32	1,04	00:11:59	3,57	4423
d198	15780	15948	15948,00	15948	15948	5,93	1,88	35,16	00:28:02	1,06	10046
kroB200	29437	29795	30301,00	30301	30301	1940,40	613,61	3765142,64	00:27:15	1,22	9547
gil262	2378	2468	2504,00	2504	2504	149,42	47,25	22326,53	00:16:56	3,78	3429
pr264	49135	49213	49522,00	49522	49522	14050,13	4443,04	197406253,49	00:45:17	0,16	9160
a280	2579	2645	2668,00	2668	2668	0,00	0,00	0,00	00:30:17	2,56	5418
lin318	42029	43163	43922,04	43910	46979	3502,88	1107,71	12270169,23	00:21:48	2,70	2975
fl417	11861	12014	12036,00	12036	12036	6,23	1,97	38,82	01:28:21	1,29	6614
d493	35002	36523	36851,00	36851	36851	2637,61	834,09	6956999,70	01:18:27	4,35	4046
		Resultados para un experimento									
u574	36902	39595	39595,00	39595	39595	0,00	0,00	0,00	01:47:00	7,30	3972
p654	34643	35091	35091,00	35091	35091	0,00	0,00	0,00	03:10:52	1,29	5198
pr1002	259045	282126	282126,00	282126	282126	0,00	0,00	0,00	08:15:18	8,91	5439
pcb1173	56892	62353	62353,00	62353	62353	0,00	0,00	0,00	13:33:54	9,60	6337
u2152	64253	71672	71672,00	71672	71672	0,00	0,00	0,00	02:56:35	11,55	6838

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 256 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7553,06	7542	8425	100,33	31,73	10066,79	00:00:34	0,00	2704
lin105	14379	14379	14383,75	14379	15589	221,68	70,10	49141,87	00:09:03	0,00	11025
ch150	6528	6559	6560,87	6559	7037	109,42	34,60	11972,55	00:35:37	0,47	22500
rat195	2323	2387	2406,73	2406	2592	120,49	38,10	14518,06	01:43:09	2,76	38025
d198	15780	15926	16004,31	15980	16822	402,01	127,13	161608,58	01:47:35	0,93	39204
kroB200	29437	29671	30060,00	30060	30060	1335,90	422,45	1784641,97	01:53:14	0,79	40000
gil262	2378	2443	2479,35	2471	2835	102,06	32,27	10416,29	05:35:30	2,73	68644
pr264	49135	49164	49226,95	49196	53459	2761,85	873,37	7627818,12	05:44:12	0,06	69696
a280	2579	2624	2654,73	2648	3093	52,98	16,75	2806,95	07:16:41	1,74	78400
		Resultados para un experimento									
lin318	42029	43205	43265,11	43205	47060	437,98	27,37	191826,50	12:15:19	2,80	101124
fl417	11861	11987	12369,03	11987	107768	5986,31	374,14	35835927,65	38:11:08	1,06	173889
d493	35002	36575	37043,67	36575	139895	6487,05	405,44	42081811,12	80:40:03	4,49	243049

HEURÍSTICAS BÁSICAS

Inicialización: Aleatoria Tamaño de la red: 256 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)	Ciclos
		Resultados para 10 experimentos									
berlin52	7542	7542	7542,00	7542	7542	134,63	42,57	18125,57	00:00:35	0,00	2704
lin105	14379	14379	14383,98	14379	15655	45,24	14,31	2046,37	00:09:06	0,00	11025
ch150	6528	6546	6627,89	6626	7109	9,78	3,09	95,60	00:35:58	0,28	22500
rat195	2323	2364	2409,00	2409	2409	2,80	0,89	7,84	01:43:48	1,76	38025
d198	15780	15917	15955,27	15951	16498	44,66	14,12	1994,23	01:48:31	0,87	39204
kroB200	29437	29774	29939,00	29939	29939	68,61	21,70	4707,72	02:02:06	1,14	40000
gil262	2378	2447	2468,00	2468	2468	3,32	1,05	11,01	05:38:54	2,90	68644
pr264	49135	49160	49189,00	49189	49189	140,19	44,33	19653,85	05:45:32	0,05	69696
a280	2579	2621	2629,70	2629	2807	7,13	2,26	50,89	07:18:26	1,63	78400
		Resultados para un experimento									
lin318	42029	43420	43420,00	43420	43420	0,00	0,00	0,00	12:21:56	3,31	101124
fl417	11861	12017	12018,27	12017	12342	20,31	1,27	412,60	38:31:36	1,32	173889
d493	35002	36444	36444,00	36444	36444	0,00	0,00	0,00	80:07:53	4,12	243049

5.3.2.- Heurísticas básicas con constructor el vecino más cercano

El caso es idéntico al anterior con la salvedad de que el constructor del camino inicial es el heurístico voraz el vecino más cercano.

El único motivo de separarlo es para clarificar la lectura. Se va a presentar la misma información: fichero de configuración de PeerSim, esquema de red y las tablas de resultados.

Al final del apartado presentaremos algunas gráficas que compararán ambos experimentos.

```
CYCLES 5000
LENGTH 280
NETSIZE 32
CACHE 30
K 28

random.seed 123456789
simulation.cycles CYCLES
network.size NETSIZE
network.node peersim.core.GeneralNode

# Initializers -----------------

# Linkable initializer
init.freeba WireScaleFreeBA
init.freeba.protocol lnk
init.freeba.k K

init.etsp FileETSP
init.etsp.file /home/paco/a280.tsp
init.etsp.protocol problem

init.constructor NNHProtocol
init.constructor.instance problem
init.constructor.size LENGTH
init.constructor.protocol improve

init.mininit MinTourInitializer
init.mininit.size LENGTH
init.mininit.protocol min
init.mininit.source improve

include.init freeba etsp constructor mininit
```

```
# Protocols ------------------------------

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache CACHE

protocol.problem TSPHolder

protocol.improve TwoOpt
protocol.improve.instance problem
protocol.improve.size LENGTH

protocol.min MinTourHolder
protocol.min.linkable lnk
protocol.min.times 10
protocol.min.protocol improve
protocol.min.instance problem

protocol.restart MinTourRestart
protocol.restart.instance problem
protocol.restart.protocol min
protocol.restart.improve improve
protocol.restart.max 50
protocol.restart.min 20

# Controller and Observers -------------------------

control.mino MinTourObserver
control.mino.protocol min
control.mino.file true
control.mino.filename out-nnh-2opt-a280-32-110n-10n.txt

control.minv MinTourValueObserver
control.minv.protocol min
control.minv.multiple 10

control.dnet ProbabilityNetwork
control.dnet.maxsize NETSIZE
control.dnet.multiple 10
control.dnet.init.0 ScaleFreeNI
control.dnet.init.0.protocol lnk
control.dnet.init.0.k K
control.dnet.init.1 TSPHolderNI
control.dnet.init.1.protocol problem
control.dnet.init.2 RandomImproversNI
control.dnet.init.2.protocol improve
control.dnet.init.3 MinTourNI
control.dnet.init.3.protocol min
control.dnet.init.3.improve improve
```

		Inicialización: NNH Tamaño de la red: 32 Volatilidad de la red: 1/n Tiempo sim: 10*n								
INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	303,20	95,88	91931,74	00:00:02	0,00
lin105	14379	14534	14660,00	14660	14660	2328,37	736,30	5421310,08	00:00:06	1,08
ch150	6528	6646	6688,67	6646	7286	994,99	314,64	990003,67	00:00:04	1,81
rat195	2323	2402	2433,00	2433	2433	547,37	173,09	299617,79	00:00:07	3,40
d198	15780	15977	16221,00	16221	16221	3579,72	1132,01	12814378,93	00:00:07	1,25
kroB200	29437	30484	31526,00	31526	31526	5063,33	1601,17	25637307,97	00:00:16	3,56
gil262	2378	2472	2591,41	2561	3443	670,69	212,09	449831,66	00:00:22	3,95
pr264	49135	49960	51880,00	51880	51880	17772,16	5620,05	315849651,27	00:00:17	1,68
a280	2579	2669	2775,00	2775	2775	540,26	170,85	291880,82	00:00:59	3,49
lin318	42029	44237	44969,03	44933	45978	11048,81	3493,94	122076292,92	00:00:52	5,25
fl417	11861	12131	12343,00	12343	12343	4242,41	1341,57	17998062,34	00:01:33	2,28
d493	35002	36782	37169,00	37169	37169	4281,18	1353,83	18328521,37	00:01:14	5,09
		Resultados para un experimento								
u574	36902	40232	40232,00	40232	40232	0,00	0,00	0,00	00:01:24	9,02
p654	34643	35609	44992,27	35609	316796	51335,56	9074,93	2635339823,79	00:03:36	2,79
pr1002	259045	281954	282077,71	281954	285418	654,63	115,72	428546,29	00:04:09	8,84
pcb1173	56892	62442	62505,86	62442	64294	343,91	60,79	118272,55	00:06:00	9,76
u2152	64253	69452	69452,00	69452	69452	0,00	0,00	0,00	00:21:05	8,09

Inicialización: NNH Tamaño de la red: 32 Volatilidad de la red: 1/10n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	377,41	119,35	142436,39	00:00:05	0,00
lin105	14379	14535	14577,00	14577	14577	2305,51	729,07	5315379,48	00:00:09	1,08
ch150	6528	6646	6646,00	6646	6646	765,38	242,03	585800,89	00:00:05	1,81
rat195	2323	2402	2433,00	2433	2433	533,01	168,55	284095,62	00:00:08	3,40
d198	15780	16044	16044,00	16044	16044	4549,13	1438,56	20694611,85	00:00:21	1,67
kroB200	29437	30700	30971,81	30778	36980	3563,78	1126,97	12700515,72	00:00:29	4,29
gil262	2378	2472	2524,00	2524	2524	412,17	130,34	169882,99	00:00:26	3,95
pr264	49135	49912	51239,00	51239	51239	15909,33	5030,97	253106759,73	00:00:34	1,58
a280	2579	2669	2689,00	2689	2689	510,47	161,42	260576,95	00:00:53	3,49
lin318	42029	44102	44280,00	44280	44280	12562,40	3972,58	157813979,37	00:00:26	4,93
fl417	11861	12097	12108,00	12108	12108	5096,04	1611,51	25969619,98	00:01:46	1,99
d493	35002	36736	37206,00	37206	37206	2621,20	828,90	6870689,89	00:01:05	4,95
		Resultados para un experimento								
u574	36902	40232	40232,00	40232	40232	0,00	0,00	0,00	00:01:22	9,02
p654	34643	35740	35740,00	35740	35740	0,00	0,00	0,00	00:03:23	3,17
pr1002	259045	281954	281954,00	281954	281954	0,00	0,00	0,00	00:04:39	8,84
pcb1173	56892	62442	62782,45	62442	72996	0,00	0,00	0,00	00:06:57	9,76
u2152	64253	69452	69716,71	69452	77658	1473,84	260,54	2172207,61	00:22:03	8,09

HEURÍSICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 32 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	326,10	103,12	106338,24	00:00:06	0,00
lin105	14379	14379	14412,00	14412	14412	2161,90	683,65	4673795,25	00:01:16	0,00
ch150	6528	6622	6704,74	6646	8467	1441,35	455,80	2077496,87	00:05:21	1,44
rat195	2323	2402	2433,00	2433	2433	482,27	152,51	232580,04	00:15:05	3,40
d198	15780	15977	17315,31	16034	56020	3727,44	1178,72	13893787,51	00:16:08	1,25
kroB200	29437	29976	30154,00	30154	30154	5415,83	1712,64	29331228,54	00:16:37	1,83
gil262	2378	2466	2584,67	2509	2712	835,64	264,25	698296,20	00:56:09	3,70
pr264	49135	49368	50334,16	49831	55198	22757,72	7196,62	517913691,38	00:50:15	0,47
a280	2579	2662	2672,17	2663	2938	638,63	201,95	407843,53	01:04:33	3,22
		Resultados para un experimento								
lin318	42029	43499	44044,19	43518	45557	906,95	160,33	822563,29	01:48:45	3,50
fl417	11861	12021	12278,03	12021	13395	450,22	79,59	202696,03	05:42:24	1,35
d493	35002	36661	36694,44	36661	37731	189,15	33,44	35778,12	11:47:40	4,74
u574	36902	39385	39592,03	39385	40710	488,80	86,41	238920,74	22:04:12	6,73

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 32 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Es	Error Estánda	Varianza Empí	Tiempo	Error
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	316,68	100,14	100289,17	00:00:06	0,00
lin105	14379	14379	14434,00	14434	14434	2221,66	702,55	4935778,30	00:01:20	0,00
ch150	6528	6624	6661,31	6624	7221	871,31	275,53	759183,61	00:05:29	1,47
rat195	2323	2402	2443,88	2433	2607	483,95	153,04	234203,75	00:15:57	3,40
d198	15780	15971	16037,31	16025	16419	4181,45	1322,29	17484523,66	00:16:33	1,21
kroB200	29437	29956	30320,00	30320	30320	5481,59	1733,43	30047870,29	00:17:21	1,76
gil262	2378	2464	2494,00	2494	2494	604,06	191,02	364894,46	00:52:53	3,62
pr264	49135	49498	49898,00	49898	49898	20029,57	6333,91	401183659,31	00:52:51	0,74
a280	2579	2655	2655,00	2655	2655	568,49	179,77	323181,70	01:07:06	2,95
		Resultados para un experimento								
lin318	42029	43655	43837,32	43655	46481	705,74	124,76	498068,82	01:54:17	3,87
fl417	11861	12017	12063,50	12017	13505	263,04	46,50	69192,00	05:55:41	1,32
d493	35002	36522	36522,00	36522	36522	0,00	0,00	0,00	12:12:27	4,34
u574	36902	39390	39390,00	39390	39390	0,00	0,00	0,00	23:06:29	6,74

Inicialización: NNH Tamaño de la red: 64 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7615,42	7542	11220	437,66	138,40	191548,17	00:00:05	0,00
lin105	14379	14434	15016,68	14578	39401	2705,03	855,41	7317178,82	00:00:10	0,38
ch150	6528	6646	6675,84	6646	7556	1004,39	317,62	1008802,09	00:00:18	1,81
rat195	2323	2402	2436,89	2433	2670	557,28	176,23	310559,58	00:00:30	3,40
d198	15780	15976	16161,70	16117	17474	3370,66	1065,90	11361348,62	00:02:02	1,24
kroB200	29437	30222	30832,80	30760	32944	4688,62	1482,67	21983128,18	00:01:36	2,67
gil262	2378	2472	2525,80	2524	2632	645,79	204,22	417047,76	00:01:10	3,95
pr264	49135	49800	50207,21	49995	54770	16053,95	5076,71	257729443,63	00:03:16	1,35
a280	2579	2648	2697,29	2690	3120	458,26	144,91	210000,59	00:02:18	2,68
lin318	42029	43458	43543,53	43458	46024	12145,70	3840,81	147518134,45	00:04:13	3,40
fl417	11861	12034	12118,16	12094	12843	5630,62	1780,56	31703828,74	00:05:21	1,46
d493	35002	36674	36935,92	36674	49505	3947,98	1248,46	15586541,48	00:08:19	4,78
		Resultados para un experimento								
u574	36902	40063	40175,87	40063	42161	447,99	56,00	200693,07	00:06:26	8,57
p654	34643	35266	35308,61	35266	3786	332,77	41,60	110734,44	00:22:14	1,80
pr1002	259045	281954	282509,31	281954	294895	2441,49	305,19	5960894,18	00:16:45	8,84
pcb1173	56892	62442	62717,10	62442	72113	1380,45	172,56	1905644,02	00:24:43	9,76
u2152	64253	69452	69520,90	69452	73448	524,70	65,59	275310,62	01:21:28	8,09

Inicialización: NNH Tamaño de la red: 64 Volatilidad de la red: 1/10n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 expermientos								
berlin52	7542	7542	7542,00	7542	7542	355,19	112,32	126163,45	00:00:05	0,00
lin105	14379	14379	14670,70	14579	20356	2248,61	711,07	5056262,83	00:00:27	0,00
ch150	6528	6619	6670,52	6646	8191	972,65	307,58	946038,38	00:00:19	1,39
rat195	2323	2402	2433,00	2433	2433	434,45	137,39	188751,04	00:00:32	3,40
d198	15780	16096	16144,00	16144	16144	3251,67	1028,27	10573377,46	00:01:43	2,00
kroB200	29437	30202	30479,00	30479	30479	5531,31	1749,15	30595369,98	00:01:52	2,60
gil262	2378	2472	2507,00	2507	2507	696,59	220,28	485239,47	00:02:41	3,95
pr264	49135	49588	50271,67	50198	54839	16226,99	5131,42	263315067,53	00:02:57	0,92
a280	2579	2669	2693,00	2693	2693	481,34	152,21	231684,73	00:02:20	3,49
lin318	42029	43795	44375,00	44375	44375	11038,95	3490,82	121858348,30	00:02:40	4,20
fl417	11861	12100	12143,00	12143	12143	4296,39	1358,64	18458971,87	00:09:20	2,02
d493	35002	36736	37069,51	37051	38217	4397,10	1390,49	19334512,24	00:08:39	4,95
		Resultados para un experimento								
u574	36902	39875	39875,00	39875	39875	0,00	0,00	0,00	00:12:49	8,06
p654	34643	35233	35233,00	35233	35233	0,00	0,00	0,00	00:12:30	1,70
pr1002	259045	281954	282693,56	281954	328546	5870,04	733,76	34457372,44	00:18:23	8,84
pcb1173	56892	62442	62609,52	62442	72996	1329,68	166,21	1768046,29	00:30:26	9,76
u2152	64253	69452	69580,22	69452	77658	1025,75	128,22	1052163,06	01:30:18	8,09

Inicialización: NNH Tamaño de la red: 64 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	436,90	138,16	190881,61	00:00:10	0,00
lin105	14379	14379	14394,60	14379	15346	2143,53	677,84	4594725,40	00:02:35	0,00
ch150	6528	6623	6652,75	6646	7078	1058,99	334,88	1121450,26	00:10:36	1,46
rat195	2323	2402	2453,10	2432	2650	608,15	192,31	369843,32	00:30:53	3,40
d198	15780	15941	16076,15	16002	16792	4121,03	1303,19	16982917,58	00:32:29	1,02
kroB200	29437	29884	30277,64	30021	32666	5786,03	1829,70	33478144,15	00:37:43	1,52
gil262	2378	2462	2495,71	2475	2696	761,55	240,82	579954,98	01:41:33	3,53
pr264	49135	49262	50195,84	49612	58324	21183,30	6698,75	448732225,21	01:42:51	0,26
a280	2579	2629	2654,94	2629	2957	488,43	154,46	238564,44	02:11:30	1,94
		Resultados para un experimento								
lin318	42029	43422	45930,65	43422	176747	16771,80	2096,48	281293285,30	03:42:28	3,31
fl417	11861	12003	12057,75	12003	12587	171,57	21,45	29436,38	11:35:25	1,20
d493	35002	36467	36577,60	36467	37849	376,51	47,06	141763,52	24:31:46	4,19
u574	36902	39281	39351,98	39281	41517	395,17	49,40	156160,50	40:44:37	6,45

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 64 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7596,00	7596	7596	441,87	139,73	195251,40	00:00:11	0,00
lin105	14379	14379	14401,00	14401	14401	1943,38	614,55	3776732,29	00:02:39	0,00
ch150	6528	6598	6633,00	6633	6633	949,06	300,12	900710,07	00:10:55	1,07
rat195	2323	2398	2428,69	2425	2661	584,24	184,75	341333,29	00:31:42	3,23
d198	15780	15946	15991,00	15991	15991	3731,91	1180,13	13927149,98	00:33:48	1,05
kroB200	29437	29835	29971,23	29927	32758	4880,84	1543,46	23822641,79	00:35:19	1,35
gil262	2378	2472	2475,44	2472	2582	634,03	200,50	401994,59	03:20:33	3,95
pr264	49135	49338	49629,00	49629	49629	17708,19	5599,82	313579916,04	01:45:58	0,41
a280	2579	2638	2666,00	2666	2666	674,40	213,26	454815,73	02:14:37	2,29
		Resultados para un experimento								
lin318	42029	43437	43482,47	43437	46347	363,75	45,47	132314,06	03:48:49	3,35
fl417	11861	12033	12033,00	12033	12033	0,00	0,00	0,00	13:11:26	1,45
d493	35002	36705	36705,00	36705	36705	0,00	0,00	0,00	25:19:06	4,87
u574	36902	39268	39268,00	39268	39268	0,00	0,00	0,00	46:11:44	6,41

		Inicialización: NNH Tamaño de la red: 128 Volatilidad de la red: 1/n Tiempo sim: 10*n								
INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7568,10	7542	8403	432,18	136,67	186775,91	00:00:11	0,00
lin105	14379	14379	14483,05	14459	17489	2096,50	662,97	4395325,04	00:00:52	0,00
ch150	6528	6625	6658,34	6646	7514	1096,93	346,88	1203260,93	00:01:13	1,49
rat195	2323	2402	2434,02	2433	2560	633,08	200,20	400791,31	00:02:28	3,40
d198	15780	15976	15988,19	15976	16484	4133,17	1307,02	17083062,52	00:05:29	1,24
kroB200	29437	30176	30285,32	30223	34318	4768,76	1508,02	22741094,63	00:05:47	2,51
gil262	2378	2468	521,72	2515	2718	675,67	213,67	456529,37	00:07:16	3,78
pr264	49135	49632	49728,40	49689	54614	24666,27	7800,16	608425008,54	00:06:45	1,01
a280	2579	2657	2717,40	2709	2952	493,88	156,18	243913,21	00:05:42	3,02
lin318	42029	43315	43495,08	43448	46794	11225,78	3549,90	126018130,55	00:08:27	3,06
fl417	11861	12014	12075,03	12014	14564	5361,36	1695,41	28744158,67	00:37:38	1,29
d493	35002	36736	36985,51	36980	37206	3828,27	1210,61	14655669,71	00:20:48	4,95
		Resultados para un experimento								
u574	36902	39664	39692,07	39664	41658	223,11	19,72	49780,29	00:23:20	7,48
p654	34643	35226	35328,74	35226	48068	1148,62	101,52	1319335,71	01:09:02	1,68
pr1002	259045	281211	281740,28	281211	298814	2373,32	209,77	5632633,21	02:05:30	8,56
pcb1173	56892	62442	62478,97	62442	64771	292,25	25,83	85410,27	01:39:21	9,76
u2152	64253	69452	69452,00	69452	69452	0,00	0,00	0,00	05:55:40	8,09

		Inicialización: NNH Tamaño de la red: 128 Volatilidad de la red: 1/10n Tiempo sim: 10*n								
INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	323,64	102,35	104745,36	00:00:13	0,00
lin105	14379	14379	14379,00	14379	14379	1916,62	606,09	3673433,29	00:02:21	0,00
ch150	6528	6608	6646,00	6646	6646	764,34	241,71	584222,82	00:01:08	1,23
rat195	2323	2402	2433,00	2433	2433	443,51	140,25	196703,94	00:02:24	3,40
d198	15780	15984	16035,31	16014	18720	3290,18	1040,45	10825275,26	00:06:43	1,29
kroB200	29437	30160	30384,00	30384	30384	5716,65	1807,76	32680096,45	00:07:20	2,46
gil262	2378	2472	2487,00	2487	2487	545,13	172,39	297169,71	00:09:51	3,95
pr264	49135	49335	49335,00	49335	49335	16661,40	5268,80	277602335,29	00:07:56	0,41
a280	2579	2652	2659,00	2659	2659	569,93	180,23	324819,66	00:12:45	2,83
lin318	42029	43436	43810,00	43810	43810	7409,36	2343,05	54898641,73	00:08:51	3,35
fl417	11861	12038	12080,00	12080	12080	4465,37	1412,07	19939506,41	00:22:01	1,49
d493	35002	36694	37071,00	37071	37071	130,51	41,27	17032,97	00:16:48	4,83
		Resultados para 1 experimento								
u574	36902	39726	39726,00	39726	39726	0,00	0,00	0,00	00:28:14	7,65
p654	34643	35078	35078,00	35078	35078	0,00	0,00	0,00	00:57:57	1,26
pr1002	259045	281954	281954,00	281954	281954	0,00	0,00	0,00	01:13:14	8,84
pcb1173	56892	62442	62442,00	62442	62442	0,00	0,00	0,00	01:41:24	9,76
u2152	64253	69452	69452,00	69452	69452	0,00	0,00	0,00	05:46:25	8,09

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 128 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7561,13	7542	7841	427,26	135,11	182552,45	00:02:06	0,00
lin105	14379	14379	14403,66	14379	15730	2320,26	733,73	5383588,79	00:05:13	0,00
ch150	6528	6558	6593,00	6593	6593	978,81	309,53	958066,90	00:21:31	0,46
rat195	2323	2393	2432,30	2397	3454	540,93	171,06	292607,84	00:00:00	3,01
d198	15780	15909	15949,36	15909	16804	4249,77	1343,90	18060542,36	01:05:39	0,82
kroB200	29437	29841	29924,67	29876	31920	5499,68	1739,15	30246513,24	01:08:47	1,37
gil262	2378	2463	2489,26	2476	2692	648,97	205,22	421161,62	03:26:53	3,57
pr264	49135	49224	49665,24	49379	67828	21780,18	6887,50	474376132,98	03:29:59	0,18
a280	2579	2637	2668,23	2661	2855	698,44	220,87	487823,50	04:25:35	2,25
		Resultados para un experimento								
lin318	42029	43409	43670,11	43409	46699	892,86	78,92	797203,55	07:47:33	3,28
fl417	11861	12005	12069,76	12005	13688	322,61	28,51	104074,23	23:47:08	1,21
d493	35002	36545	37270,83	36545	113182	6806,64	601,63	46330401,18	49:53:17	4,41

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 128 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	355,24	112,34	126196,55	00:00:00	0,00
lin105	14379	14379	14379,00	14379	14379	2303,56	728,45	5306368,95	00:05:16	0,00
ch150	6528	6582	6623,16	6621	6898	938,33	296,73	880470,41	00:24:30	0,83
rat195	2323	2402	2409,00	2409	2409	506,09	160,04	256122,37	01:04:14	3,40
d198	15780	15897	15903,02	15897	16668	3479,68	1100,37	12108169,88	01:08:40	0,74
kroB200	29437	29784	30006,00	30006	30006	6026,46	1905,73	36318232,14	01:09:37	1,18
gil262	2378	2454	2483,20	2481	2763	648,63	205,12	420726,02	03:27:39	3,20
pr264	49135	49217	49403,73	49335	54615	19509,79	6169,54	380631993,73	03:32:11	0,17
a280	2579	2630	2644,00	2644	2644	711,89	225,12	506792,30	04:29:45	1,98
		Resultados para un experimento								
lin318	42029	43474	43474,00	43474	43474	0,00	0,00	0,00	07:39:01	3,44
fl417	11861	12012	12019,72	12012	13000	87,33	7,72	7626,12	23:47:31	1,27
d493	35002	36607	36636,62	36607	38488	235,11	20,78	55276,86	50:52:32	4,59

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 256 Volatilidad de la red: 1/n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7545,38	7542	8019	375,87	118,86	141280,85	00:00:44	0,00
lin105	14379	14379	14475,80	14379	34665	2498,42	790,07	6242094,39	00:04:02	0,00
ch150	6528	6562	6647,76	6646	7092	923,05	291,89	852016,46	00:05:09	0,52
rat195	2323	2394	2426,50	2423	2670	499,79	158,05	249793,37	00:15:39	3,06
d198	15780	15927	15951,90	15927	19917	3290,88	1040,67	10829888,40	00:17:33	0,93
kroB200	29437	29870	30239,40	30217	32348	5011,47	1584,76	25114800,84	00:16:03	1,47
gil262	2378	2459	2461,27	2460	2659	609,42	192,71	371390,08	00:30:15	3,41
pr264	49135	49415	49453,24	49417	54530	17473,40	5525,57	305319593,02	00:30:12	0,57
a280	2579	2652	2658,00	2657	2913	550,04	173,94	302544,28	00:32:53	2,83
lin318	42029	43470	43605,88	43593	45481	6073,71	1920,68	36890013,84	00:30:02	3,43
fl417	11861	11955	12079,49	12078	12456	4302,90	1360,70	18514908,84	01:14:43	0,79
d493	35002	36589	36848,58	36818	41589	797,19	252,10	635519,15	01:40:18	4,53
		Resultados para un experimento								
u574	36902	39617	39617,00	39617	39617	0,00	0,00	0,00	01:34:45	7,36
p654	34643	35086	35140,12	35086	38474	344,17	21,51	118452,81	03:44:34	1,28
pr1002	259045	279417	279445,38	279417	286598	451,47	28,22	203821,19	05:50:57	7,86
pcb1173	56892	62442	63938,13	62442	440016	23690,99	1480,69	561262992,21	06:40:34	9,76
u2152	64253	69452	69484,35	69452	73590	363,90	22,74	132425,12	23:01:47	8,09

Inicialización: NNH Tamaño de la red: 256 Volatilidad de la red: 1/10n Tiempo sim: 10*n

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	283,99	89,81	80652,12	00:00:50	0,00
lin105	14379	14379	14416,00	14416	14416	1934,45	611,73	3742114,64	00:03:34	0,00
ch150	6528	6576	6646,00	6646	6646	787,76	249,11	620571,92	00:06:01	0,74
rat195	2323	2390	2433,00	2433	2433	423,88	134,04	179675,87	00:08:51	2,88
d198	15780	15915	15981,00	15981	15981	3354,57	1060,81	11253170,46	00:13:02	0,86
kroB200	29437	30003	30313,00	30313	30313	4610,29	1457,90	21254769,93	00:15:28	1,92
gil262	2378	2477	2509,00	2509	2509	580,87	183,69	337407,17	00:26:32	4,16
pr264	49135	49213	49632,00	49632	49632	19733,88	6240,40	389425898,48	00:31:01	0,16
a280	2579	2643	2665,00	2665	2665	463,19	146,47	214544,32	00:34:20	2,48
lin318	42029	43347	43347,00	43347	43347	4997,08	1580,22	24970824,41	00:45:21	3,14
fl417	11861	11977	12032,00	12032	12032	3153,42	997,20	9944067,52	01:23:28	0,98
d493	35002	36631	36796,06	36782	38955	63,46	20,07	4026,98	02:06:28	4,65
		Resutados para un experimento								
u574	36902	39508	39508,00	39508	39508	0,00	0,00	0,00	02:48:50	7,06
p654	34643	35062	35062,00	35062	35062	0,00	0,00	0,00	03:24:49	1,21
pr1002	259045	281619	281619,00	281619	281619	0,00	0,00	0,00	06:03:40	8,71
pcb1173	56892	62442	62442,00	62442	62442	0,00	0,00	0,00	06:57:20	9,76
u2152	64253	69452	69452,00	69452	69452	0,00	0,00	0,00	23:27:55	8,09

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 256 Volatilidad de la red: 1/n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7543,92	7542	8032	366,28	115,83	134159,88	00:00:39	0,00
lin105	14379	14379	14384,23	14379	15299	2215,39	700,57	4907962,05	00:10:37	0,00
ch150	6528	6528	6559,39	6554	7078	944,82	298,78	892684,78	00:43:32	0,00
rat195	2323	2379	2385,06	2379	2713	592,61	187,40	351192,21	02:06:46	2,41
d198	15780	15926	15941,49	15927	16481	3330,99	1053,35	11095467,43	02:12:04	0,93
kroB200	29437	29721	30164,94	30029	39063	5207,63	1646,80	27119368,60	02:20:48	0,96
gil262	2378	2457	2469,76	2460	3966	617,10	195,14	380812,13	06:52:23	3,32
pr264	49135	49151	49269,02	49204	53953	16918,54	5350,11	286237089,26	07:00:07	0,03
a280	2579	2625	2683,40	2650	9542	583,99	184,67	341046,94	08:55:51	1,78
		Resultados para un experimento								
lin318	42029	43141	43227,82	43141	48277	571,27	35,70	326352,98	15:11:32	2,65
fl417	11861	12002	12055,26	12002	13511	278,99	17,44	77836,30	48:30:59	1,19
d493	35002	36435	36515,73	36435	40065	526,01	32,88	276691,00	99:15:03	4,09

HEURÍSTICAS BÁSICAS

Inicialización: NNH Tamaño de la red: 256 Volatilidad de la red: 1/10n Tiempo sim: ciudades^2

INSTANCIA	ÓPTIMO	Mínimo	Media	Mediana	Máximo	Desviación Estándar	Error Estándar	Varianza Empírica	Tiempo (hh:mm:ss)	Error (%)
		Resultados para 10 experimentos								
berlin52	7542	7542	7542,00	7542	7542	284,02	89,82	80669,36	00:00:40	0,00
lin105	14379	14379	14379,00	14379	14379	2114,58	668,69	4471460,15	00:10:31	0,00
ch150	6528	6576	6615,00	6615	6615	984,00	311,17	968253,93	00:43:48	0,74
rat195	2323	2385	2412,00	2412	2412	495,70	156,75	245715,27	02:08:50	2,67
d198	15780	15893	15983,76	15973	18728	4058,12	1283,29	16468366,22	02:13:08	0,72
kroB200	29437	29792	30002,00	30002	30002	5550,88	1755,34	30812315,67	02:19:55	1,21
gil262	2378	2452	2472,00	2472	2472	631,01	199,54	398173,15	07:10:58	3,11
pr264	49135	49198	49289,00	49289	49289	18340,56	5799,80	336376300,93	07:03:36	0,13
a280	2579	2629	2666,24	2665	2983	250,62	79,25	62810,40	09:02:09	1,94
lin318	42029	43314	43314,00	43314	43314	0,00	0,00	0,00	15:14:48	3,06
		Resultados para un experimento								
fl417	11861	11999	11999,00	11999	11999	0,00	0,00	0,00	47:44:01	1,16
d493	35002	36408	36408,00	36408	36408	0,00	0,00	0,00	103:08:28	4,02

Una vez conocidos los resultados obtenidos de estos experimentos, vamos a ver una serie de gráficas que nos permita sacar una serie de conclusiones sobre los mismos.

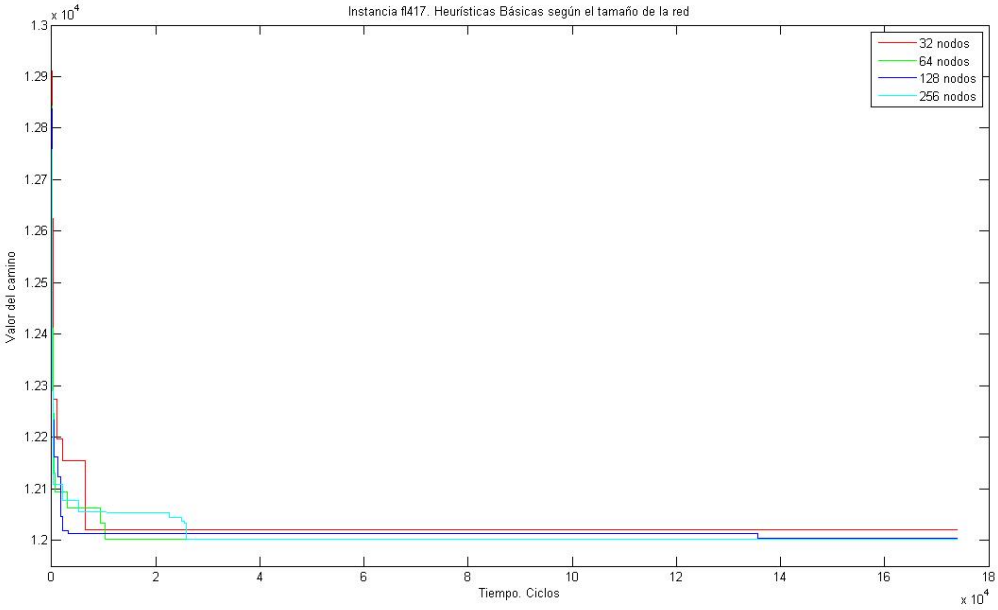


Fig. 5.4. Valor del camino para fl417 según el tamaño de la red

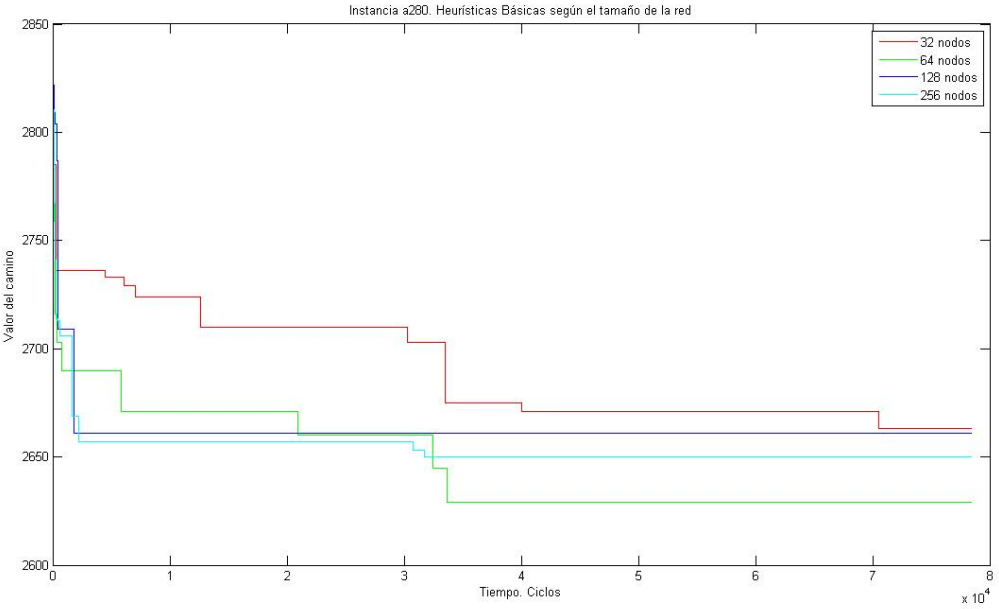


Fig. 5.5. Valor de camino para a280 según el tamaño de la red

Las dos gráficas anteriores muestran la evolución de la solución durante uno de los experimentos según el tamaño de la red. Se trata de simulaciones donde el constructor es el heurístico del vecino más cercano y para una instancia

donde la solución es aproximadamente buena –alrededor del 1% – como es fl417 y, la segunda, para una instancia donde el comportamiento es un poco peor, a280, donde el error está entorno al 3%. Se ha elegido este constructor para la elaboración de estas gráficas con el objeto de que el eje Y no tenga una escala demasiado grande y no se pueda apreciar el detalle de como van convergiendo el valor del tour conforme se va desarrollando la simulación. Lo que sí nos permite ver es el algoritmo de mejora tiene una acción muy escalonada donde hay largos periodos de tiempo donde no tiene un efecto claro. En el caso, por ejemplo, de la instancia fl417 el escalón se produce muy pronto y luego hay mejoras muy leves en algunos casos y en otros inexistentes. Esto es un síntoma de que el heurístico de construcción se queda muy próximo de un óptimo local, del cual el heurístico de mejora no logra escapar. Por este motivo, se plantean simulaciones con un constructor aleatorio, donde el heurístico de mejora parte de lugares muy distintos del espacio de soluciones.

Las gráficas siguientes muestran el error mínimo obtenido en las simulaciones de heurísticas básica cuando la volatilidad de la red es de 1/n, siendo n el número de nodos de la red. Las gráficas se presentan en parejas según el heurístico de construcción empleado y la condición de parada – esto es, se repite el mismo resultado 10*n, siendo n el número de nodos y el número de ciudades al cuadrado –. En el caso de los experimentos donde la condición de parada ha sido el número de ciudades al cuadrado, se ha ejecutado en menos instancias debido al tiempo empleado para ello.

Como podemos ver en las gráficas, en líneas generales, el resultado es mejor cuanto mayor es el tamaño de la red y cuando se emplea como camino de inicio el que proporciona el heurístico del vecino más cercano (NNH). El empleo de un inicio aleatorio como estrategia, no es más que una mera lotería, donde en algunos casos puntuales sí que existe una mejor solución.

Normalmente, es interesante y suele dar buenos resultados introducir en los heurísticos un pequeño nivel de aleatoriedad. Sin embargo, en esta caso, es demasiado alto este nivel.

Una cosa que llama la atención es un patrón que se repite en todas estas gráficas y que con independencia de los parámetros de la simulación empleados y del número de ciudades, existen instancias que tienen predisposición a obtener buenos resultados con estos heurísticos, como es el caso de *d198, pr264, fl417, p654*.

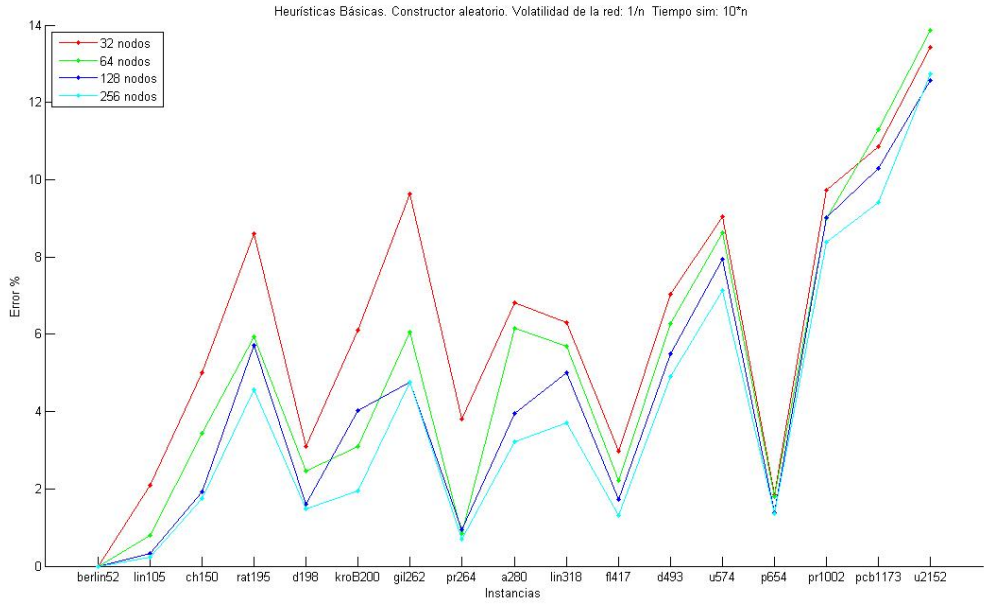


Fig. 5.6. Gráfico según tamaño de la red de heurísticas básicas con constructor aleatorio. Dinámica de red por probabilidad de 1/n. Condición de fin de simulación 10*n. Siendo n el tamaño de la red.

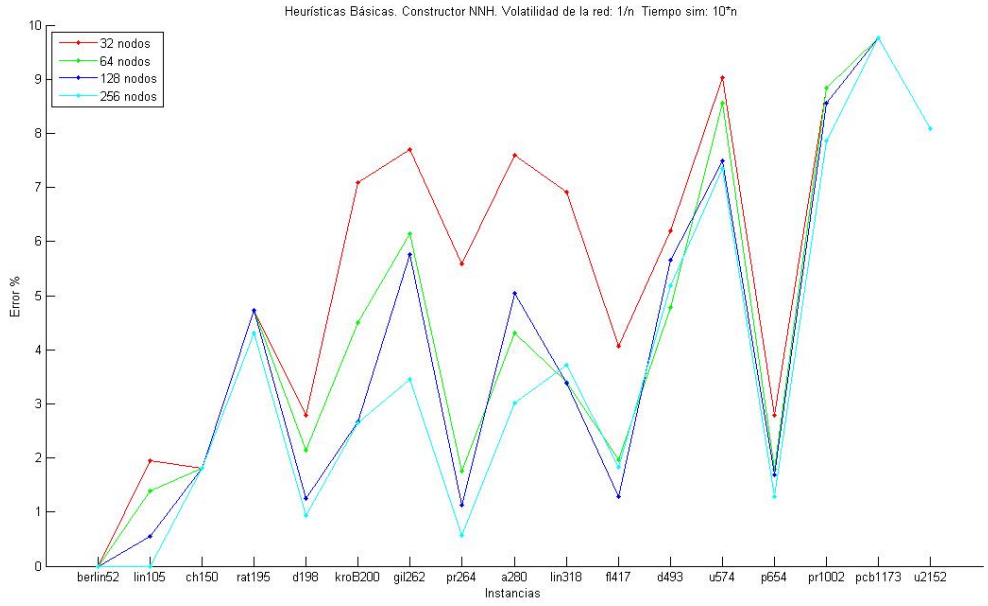


Fig 5.7. Gráfico según tamaño de la red de heurísticas básicas con constructor ávido (NNH). Dinámica de red por probabilidad de 1/n. Condición de fin de simulación 10*n. Siendo n el tamaño de la red.

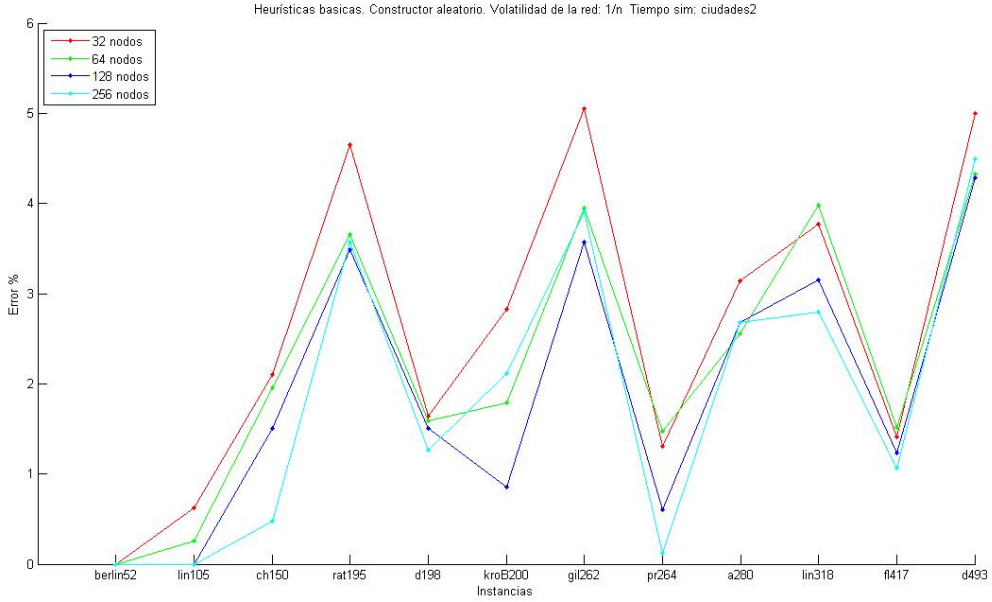


Fig. 5.8. Gráfico según tamaño de la red de heurísticas básicas con constructor aleatorio. Dinámica de red por probabilidad de 1/n. Condición de fin de simulación m^2. Siendo n el tamaño de la red y m el número de ciudades de la instancia.

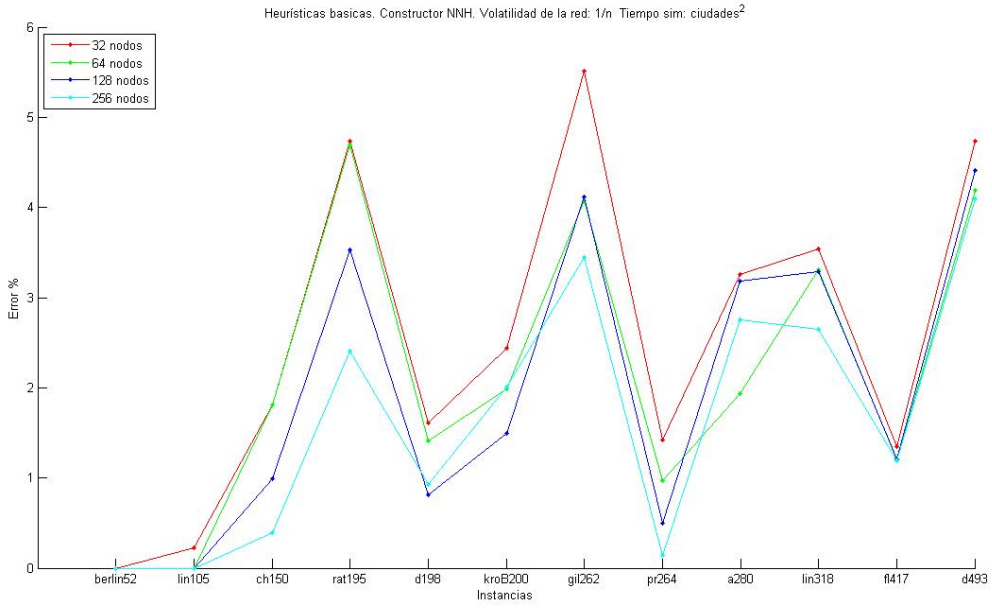


Fig. 5.9. Gráfico según tamaño de la red de heurísticas básicas con constructor voraz (NNH). Dinámica de red por probabilidad de 1/n. Condición de fin de simulación m^2. Siendo n el tamaño de la red y m el número de ciudades de la instancia.

Las dos siguientes gráficas se exponen todas las simulaciones realizadas por heurísticas básicas. Como el número de instancias no son iguales para las simulaciones que tienen como condición de finalización el repetir por 10*n el mismo valor del camino como un número de ciclos igual a m^2. Se agrupan en gráficas diferentes.

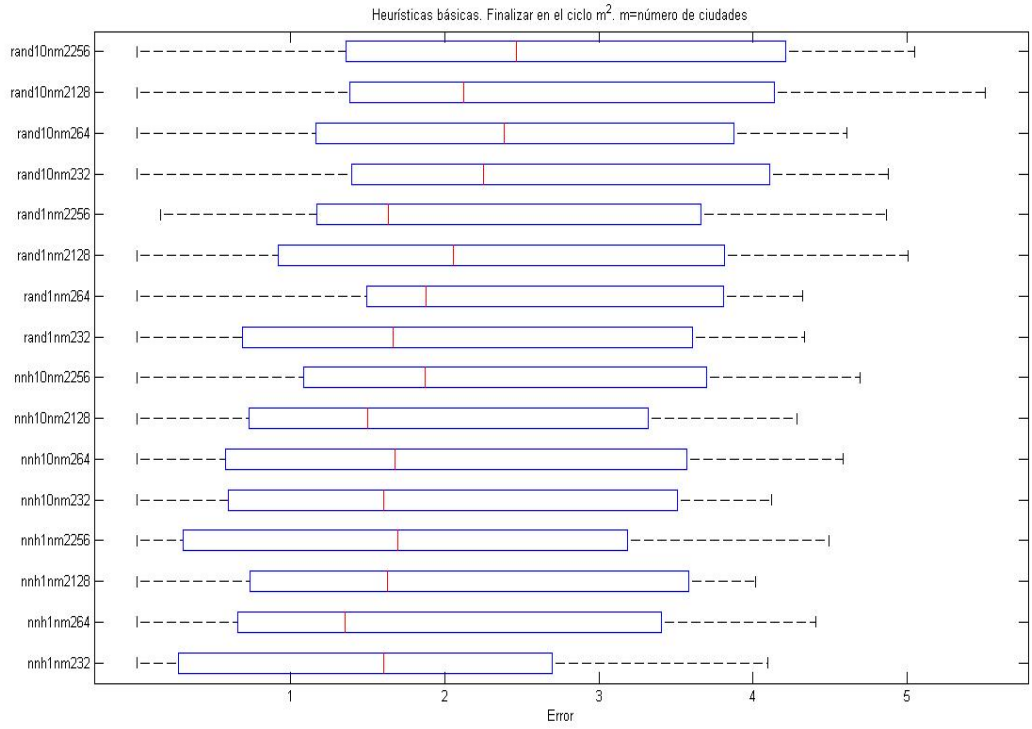


Fig. 5.10. Simulaciones con condición de finalización el número de ciclos alcance el número de ciudades^2

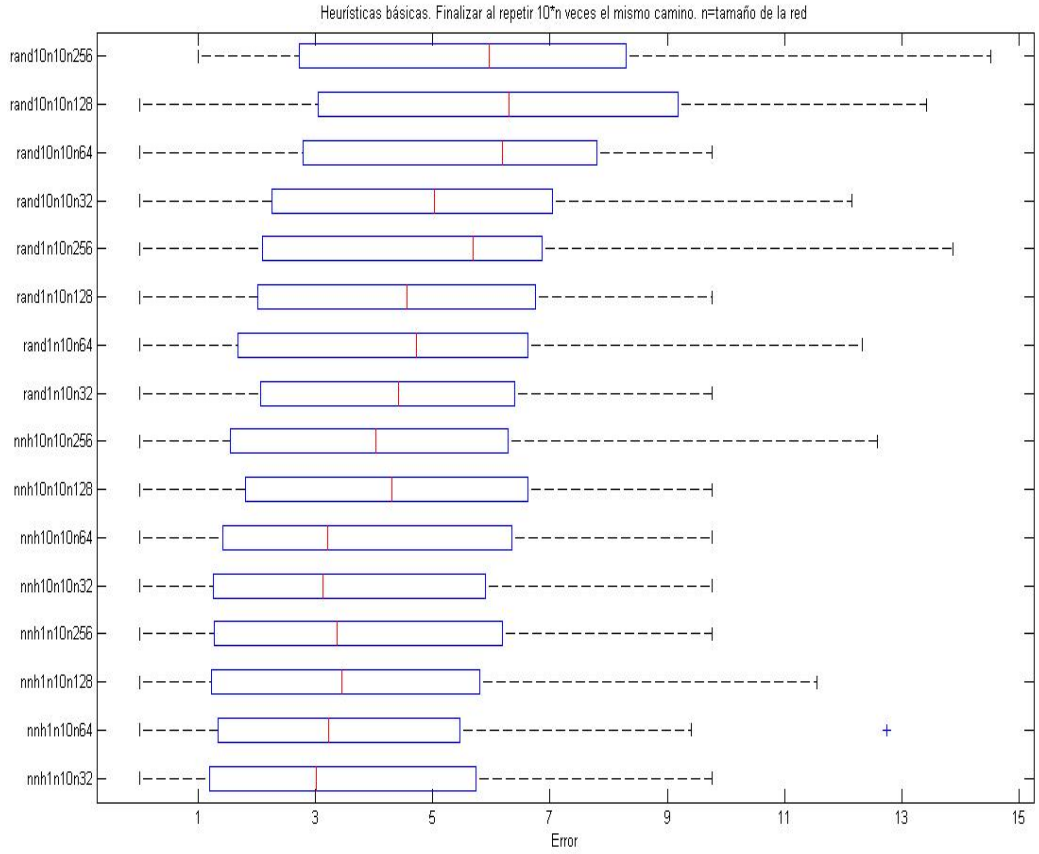


Fig. 5.11. Simulaciones con condición de finalización se repita el mismo camino durante 10*n

En las gráficas anteriores se ha usado una nomenclatura para identificar cada experimento. La cual corresponde en primer lugar al tipo de constructor empleado [rand/nnh] para el heurístico aleatorio o heurístico del vecino más cercano respectivamente; en segundo lugar la probabilidad usada en la dinámica de la red [1/*n* ó 1/10*n*], igual para la condición de parada [10*n* / *m*2] para terminar cuando se repita el mismo resultado 10* *n* veces o terminar cuando el número de ciclos sea el número de ciudades al cuadrado respectivamente. Por último, el tamaño inicial de la red [32/64/128/256]. Ponemos algunos ejemplos:

- *nnh10n10n64* → Es el experimento que se ha usado como constructor el algoritmo voraz del vecino más cercano. La dinámica de la red es con una probabilidad de 1/10*n*. La condición de parada es cuando se repita el mismo camino 10* *n* veces, donde *n* es el número de nodos de la red y con un tamaño de red inicial de 64 nodos.
- *rand1nm2128* → Es el experimento que se ha usado constructor aleatorio. La dinámica de la red es con una probabilidad de 1/*n*. La condición de parada es cuando la simulación ejecute m^2 ciclos, donde *n* es el número de nodos de la red y *m* el tamaño de la instancia. La red contiene 128 nodos iniciales.

5.3.3.- Scatter Search

Se puede decir que casi el motivo principal de este trabajo fin de grado es ver el comportamiento de este metaheurístico bioinspirado en un entorno P2P para el problema del viajante de comercio.

Esta fase de análisis ha contado con el problema de la ejecución de las grandes instancias en un entorno P2P simulado. Si bien, este problema es común a todos, en el caso de Scatter Search se acentúa porque:

- El algoritmo es mucho más elaborado que otros heurísticos aunque compartan o se aproximen a un mismo orden de complejidad computacional. Esto requiere más tiempo de CPU.
- La implementación cuenta con grandes estructuras de datos que aumentan conforme lo hace el número de ciudades de la instancia y el tamaño de la población. Esto penaliza los tiempos, al aumentar los accesos a memoria.
- Un problema inherente al propio concepto de simulación. En este caso,

una única máquina tiene que hacer el trabajo de los distintos nodos que componen la red P2P de manera secuencial.

Debido a estas causas, muchos experimentos han mostrado tiempos de ejecución elevados para los propósitos de este trabajo fin de grado. Subrayo propósitos, porque Scatter Search, en términos generales, tiene un tiempo de ejecución razonable. Sobre todo por la buena calidad de sus soluciones. Y es que, como veremos, este heurístico es capaz de solucionar instancias hasta casi el óptimo, y en muchos caso lo consigue.

El experimento que se ha llevado a cabo es la ejecución del conjunto de instancias de pruebas que hemos seleccionados (excepto u2152, por su tamaño) y probarlos con Scatter Search, en su versión mejorada, con la condición de parada de que alcance un error hasta el 1%. Para obtener una muestra lo más válida posible de resultados, cada instancia se ha ejecutado 20 veces, variando las poblaciones cada vez. Los nodos de la red compartían como recursos individuos de su población local. La red P2P es dinámica de tipo oscilante entre 16 y 32 nodos con un periodo de 500, el tamaño de la red en cada ciclo responde a las operaciones que exponemos a continuación.

$$netsize = avg + \sin\left(\frac{time \cdot \pi}{periodo}\right) \cdot ampl$$

donde *time* = el ciclo actual de la simulación

$$avg = \frac{(maxsize + minsize)}{2}$$ siendo *maxsize*=tamaño máximo de la red

$$ampl = \frac{(maxsize - minsize)}{2}$$ *minsize*=tamaño mínimo de la red

Este es unos de los fichero de configuración empleados para la ejecución de los experimentos. Se puede observar como se añade un observador para obtener datos sobre las nuevas soluciones que va generando Scatter Search en cada ciclo.

Para estas simulaciones es necesario indicar al motor el orden de ejecución de los controladores para que se obtenga un resultado correcto. Esto se ha conseguido a través de la directiva *order.control*

```
CYCLES 2000
LENGTH 52
NETSIZE 25
CACHE 10
K 10

random.seed 2134233320
simulation.cycles CYCLES

network.size NETSIZE
network.node peersim.core.GeneralNode

order.control diversity,time,newsol,sso,renewal,dnet

# Initializers ----------------

# Linkable initializer
init.freeba WireScaleFreeBA
init.freeba.protocol lnk
init.freeba.k K

init.etsp FileETSP
init.etsp.file /home/paco/berlin52.tsp
init.etsp.protocol problem

init.constructor ReferenceSetInitializer
init.constructor.instance problem
init.constructor.size LENGTH
init.constructor.population 10
init.constructor.protocol scatter

include.init freeba etsp constructor

# Protocols ------------------------------

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache CACHE

protocol.problem TSPHolder

protocol.scatter ScatterSearch
protocol.scatter.instance problem
protocol.scatter.size LENGTH

protocol.comm IndividualsTransfer
protocol.comm.linkable lnk
protocol.comm.times 5
protocol.comm.protocol scatter
protocol.comm.elements 1
```

```
# Controller and Observers -------------------------

# Order of nodes are visited

control.renewal ReferenceSetRestart
control.renewal.instance problem
control.renewal.protocol scatter
control.renewal.elements 9

control.sso ScatterSObserver
control.sso.protocol scatter
control.sso.stop true
control.sso.quality 99
control.sso.optimal 7542
control.sso.file true
control.sso.filename out-ss-q8-berlin52.txt

control.diversity EdgesTourDiversity
control.diversity.tspsize LENGTH
control.diversity.protocol scatter
control.diversity.file true
control.diversity.filename diver-ss-q8-berlin52.txt

control.newsol ScatterNewSolObserver
control.newsol.protocol scatter
control.newsol.file true
control.newsol.filename newsol-ss-q8-berlin52.txt

control.time TimeSimCounter
control.time.stop false
control.time.seconds 0

control.dnet OscillatingNetwork
control.dnet.maxsize 32
control.dnet.minsize 16
control.dnet.period 500
control.dnet.init.0 ScaleFreeNI
control.dnet.init.0.protocol lnk
control.dnet.init.0.k K
control.dnet.init.1 TSPHolderNI
control.dnet.init.1.protocol problem
control.dnet.init.2 ScatterSearchNI
control.dnet.init.2.protocol scatter
control.dnet.init.2.linkable lnk
control.dnet.init.2.population 10
```

Instancia	Óptimo	Valor tour	Error (%)	Ciclos	Nº soluciones	Tiempo (hh:mm:ss)
berlin52	7542	7542	0,00	1	13	00:00:10
lin105	14379	14379	0,00	2	96	00:00:18
ch150	6528	6562	0,53	2	196	00:01:31
rat195	2323	2342	0,82	32	169	00:27:32
d198	15780	15917	0,87	2	50	00:04:32
kroB200	29437	29644	0,70	3	57	00:06:08
gil262	2378	2389	0,71	12	122	00:23:29
pr264	49135	49338	0,41	2	36	00:04:39
a280	2579	2596	0,68	8	165	00:32:50
lin318	42029	42375	0,82	11	588	01:58:24
fl417	11861	11961	0,85	3	138	01:20:49
d493	35002	35335	0,95	42	1026	15:11:19
u574	36902	37243	0,92	34	1179	09:50:31
p654	34643	34930	0,86	2	20	00:37:49
pr1002	259045	261580	0,98	102	2654	126:50:40
pcb1173	56892	57544	0,96	82	1849	16:39:19

Fig. 5.12. Tabla de resultados del experimento para Scatter Search. Los valores expuestos son la media sobre las 20 simulaciones que se ha realizado para cada instancia del TSP. El controlador finaliza la simulación cuando el error cometido es inferior al 1%.

Al observar estos resultados en una primera pasada lo primero que se observa es la gran eficacia de Scatter Search donde en todas las instancias probadas, ha sido capaz de obtener un resultado inferior al 1% de error. Incluso en las grandes instancias.

Además de su eficacia, se observa una gran eficiencia, ya que, no necesita un número elevado de ciclos de simulación para encontrar una solución satisfactoria con los requisitos que se plantean.

Scatter Search – junto a Path Relinking, que lleva internamente – es un metaheurístico que se ha adaptado muy bien al entorno P2P. Tiene muchas posibilidades y permite una gran variedad de escenarios.

En un supuesto donde, abandonando la restricción impuesta en este experimento de parar la simulación al 1% del error, y dejando un tiempo indeterminado en una red semejante. Cabe preguntarse si Scatter Search sería capaz de encontrar el óptimo a cualquier instancia del problema del viajante de comercio que se le plantee.

Veamos, a continuación, algunas gráfica que nos arrojen más datos sobre este experimento.

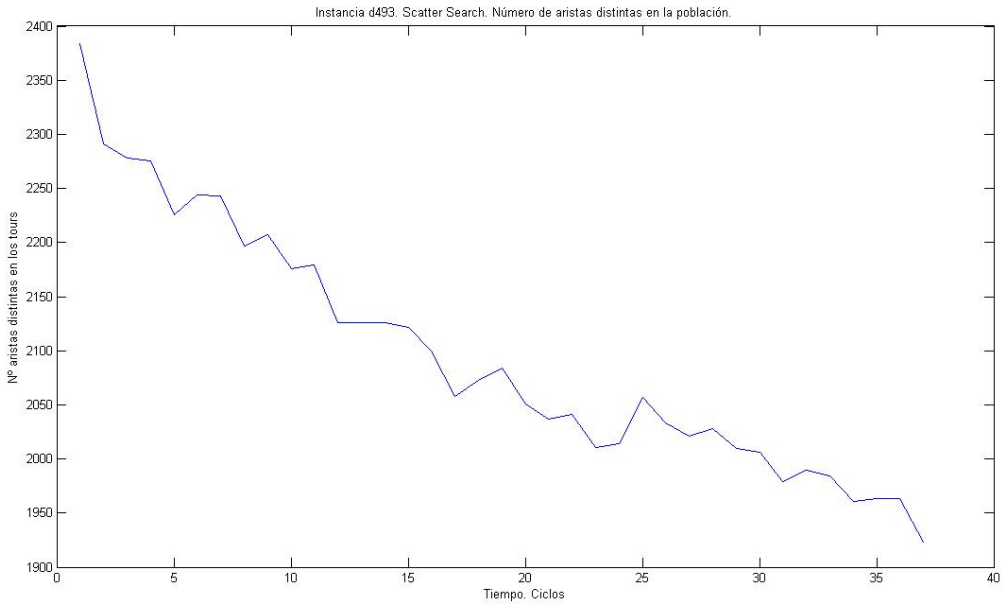


Fig. 5.13. Gráfico de diversidad de aristas para la instancia d493

El comportamiento normal la diversidad es que comience con un valor alto cuando existe una población inicial diversa. Conforme el metaheurístico va haciendo su trabajo y se aproxima a una zona del espacio de soluciones, es normal que los tours sean muy parecidos entre sí y la diversidad de aristas

disminuya con el tiempo.

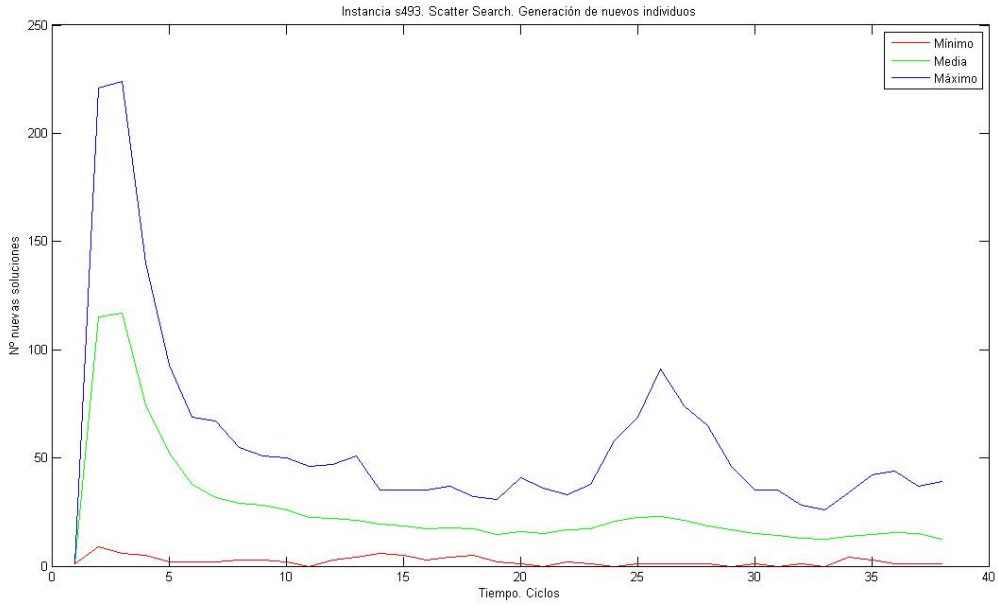


Fig. 5.14. Nuevos caminos generados en cada ciclo para instancia d493.

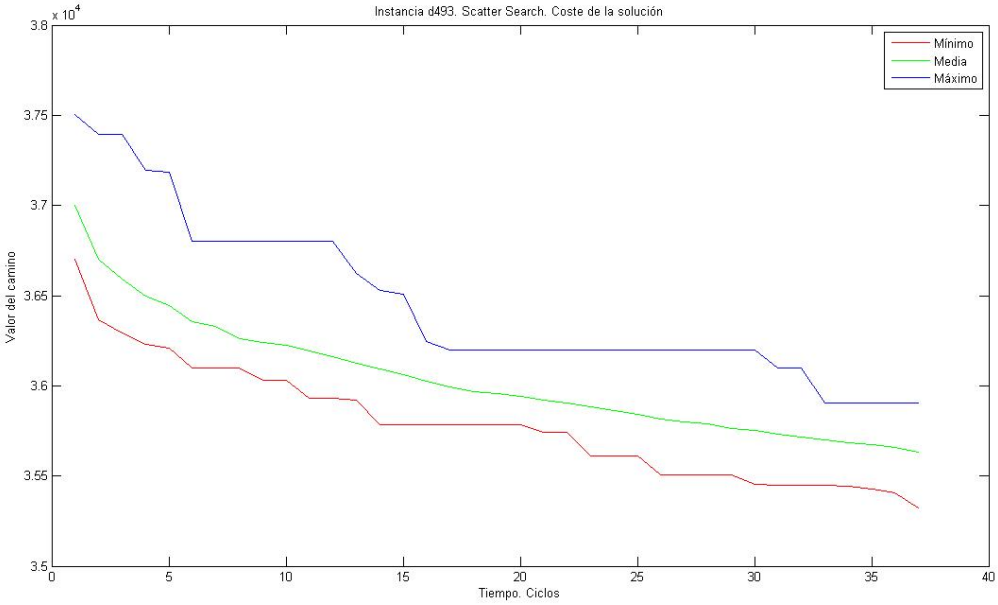


Fig. 5.15. Valores de los tour durante la simulación para instancia d493.

En cuanto a las nuevas soluciones, la línea de la gráfica más significativa puede se la media, puesto que el máximo estará expuesto a variaciones importantes cuando ocurra la fase de reinicio del conjunto de referencia, en la cual se hace un aporte extra de individuos a la población.

Las siguientes gráficas corresponden a las instancias “pr1002” y “pdc1173”.

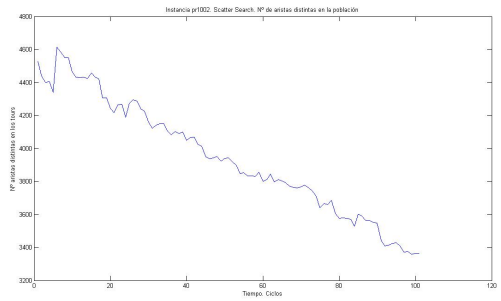


Fig. 5.16. Gráfico de diversidad de aristas para la instancia pr1002

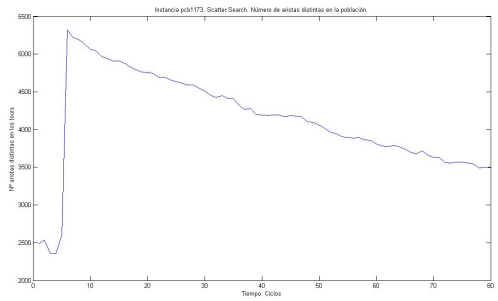


Fig. 5.17. Gráfico de diversidad de aristas para la instancia pcb1173

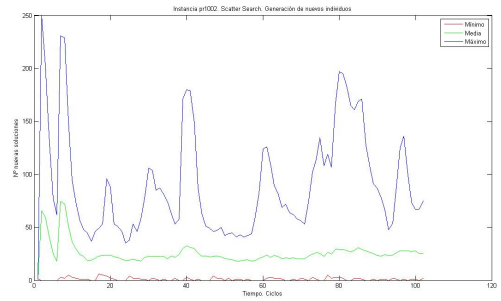


Fig. 5.18. Nuevos caminos generados en cada ciclo para instancia pr1002.

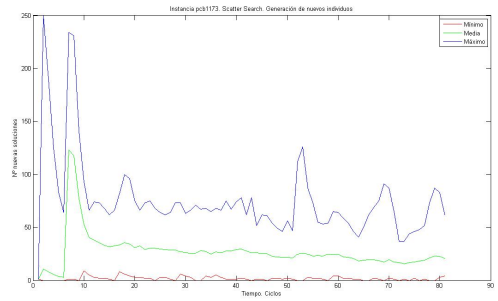


Fig. 5.19. Nuevos caminos generados en cada ciclo para instancia pcb1173.

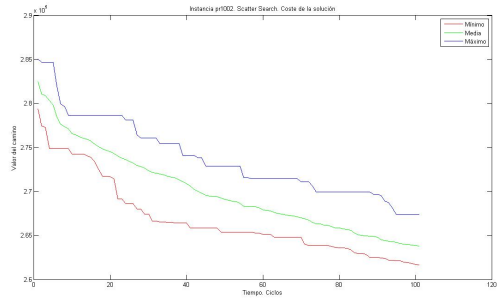


Fig. 5.20. Valores de los tour durante la simulación para instancia pr1002.

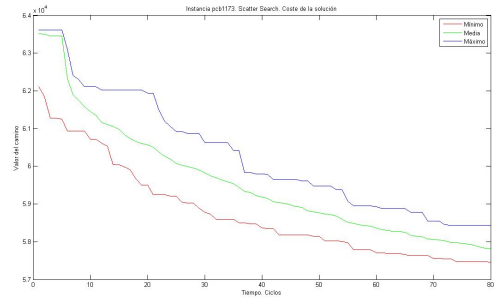


Fig. 5.21. Valores de los tour durante la simulación para instancia pcb1173.

Las gráficas de nuevas soluciones irremediablemente parten de un valor 0 ya que al inicio no se generan nuevas soluciones por el heurístico, si no que existe unos caminos iniciales.

Salvando este punto inicial, podemos ver como la media de nuevas soluciones y la gráfica de diversidad guardan un cierto parecido, sobre todo en los picos y en la primera mitad del tiempo. La principal razón para ello es que el algoritmo en el inicio explora una gran región del espacio de soluciones. Esta situación puede dar origen a que las nuevas soluciones que se aportan son muy distintas entre sí y, en consecuencia, las aristas de estos caminos forman conjuntos disjuntos.

Conforme avanza la simulación, y como hemos mencionado antes, Scatter Search se va aproximando a un óptimo (local o global) la diversidad de aristas disminuye considerablemente, ya que, los caminos que formen parte de la población serán muy cercanos entre sí. Por lo tanto, aunque se añadan nuevas soluciones, la diversidad disminuye por propio efecto del algoritmo.

En cuanto a las gráficas de los valores de los caminos. Se puede observar su decrecimiento paulatino conforme avanza la ejecución. Muy distinto a lo que hemos visto en heurísticas básicas donde existe un escalonado rápido y se mantiene en el tiempo prácticamente el mismo valor. Este es uno de los motivos por el cual se ha afirmado que Scatter Search se ha adaptado muy bien a un entorno P2P. Se evidencia que la colaboración entre nodos es muy fructífera en Scatter Search, ya que, el algoritmo por sí solo (fuera de una red P2P) no tiene una curva decreciente tan suave.

5.3.4.- Búsqueda Tabú

Los experimentos con Búsqueda Tabú (*TS, Tabu Search*) han sido menores que con el resto. Esto es debido principalmente a que sus resultados no ha sido todo lo prometedores que se esperaba. Se ha usado una red oscilante con un número inicial de nodos de 64 cuyo máximo son 75 nodos y un mínimo de 55 con un periodo de 500. El tamaño de la red en cada ciclo se corresponde con las operaciones que se detallan en el apartado anterior de Búsqueda Dispersa. La duración de la simulación es el número de ciudades de la instancia al cuadrado. Es un tiempo muy largo, pero como se ha observado que en algunas instancias el algoritmo realizaba leves mejoras conforme avanzaba el tiempo, se ha intentado dejar un margen bastante amplio para su acción.

Veamos uno de los ficheros de configuración empleados.

```
CYCLES 78400
LENGTH 280
NETSIZE 64
CACHE 60
K 55

random.seed 1234567890
simulation.cycles CYCLES

network.size NETSIZE
network.node peersim.core.GeneralNode
```

```
# Initializers -----------------

# Linkable initializer
init.freeba WireScaleFreeBA
init.freeba.protocol lnk
init.freeba.k K

init.etsp FileETSP
init.etsp.file /home/paco/a280.tsp
init.etsp.protocol problem

init.constructor NNHProtocol
init.constructor.instance problem
init.constructor.size LENGTH
init.constructor.protocol tabu

init.mininit MinTourInitializer
init.mininit.size LENGTH
init.mininit.protocol min

include.init freeba etsp constructor

# Protocols ------------------------------

protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache CACHE

protocol.problem TSPHolder

protocol.tabu TabuSearch
protocol.tabu.instance problem
protocol.tabu.size LENGTH

protocol.min MinTourHolder
protocol.min.linkable lnk
protocol.min.times 50
protocol.min.protocol tabu

# Controller and Observers -------------------------

control.sso TabuSObserver
control.sso.protocol min
control.sso.file true
control.sso.filename out-tabu-a280.txt
```

```
control.time TimeSimCounter
control.dnet OscillatingNetwork
control.dnet.maxsize 75
control.dnet.minsize 55
control.dnet.period 500
control.dnet.init.0 ScaleFreeNI
control.dnet.init.0.protocol lnk
control.dnet.init.0.k K
control.dnet.init.1 TSPHolderNI
control.dnet.init.1.protocol problem
control.dnet.init.2 TabuSearchNI
control.dnet.init.2.protocol tabu
control.dnet.init.2.linkable lnk
control.dnet.init.3 MinTourNI
control.dnet.init.3.protocol min
control.dnet.init.3.improve tabu
```

La tabla de resultados y el esquema de red de la simulación usado son los siguientes:

Instancia	Óptimo	Error (%)	Valor tour	Ciclos	Tiempo (hh:mm:ss)
berlin52	7542	0,00	7542	2704	00:00:47
lin105	14379	8,14	15549	11025	00:14:05
ch150	6528	8,27	7068	22500	00:54:33
rat195	2323	2,93	2391	38025	01:52:12
d198	15780	6,95	16876	39204	02:39:50
kroB200	29437	7,78	31727	40000	03:14:46
gil262	2378	6,01	2521	68644	06:24:47
pr264	49135	7,34	52741	69696	08:29:03
a280	2579	5,70	2726	78400	08:22:56
lin318	42029	2,08	42903	101124	18:47:51
fl417	11861	19,43	14166	173889	20:30:40

Estos resultados, como hemos comentado, no son realmente los esperados para Búsqueda Tabú. En algunas instancias sí se ve un comportamiento aceptable en cuanto a calidad. Respecto al tiempo de ejecución, podemos decir que Búsqueda Tabú es un método bastante rápido, teniendo en cuenta la gran cantidad de ciclos de los que constaba cada experimento. En defensa de esta implementación podemos mencionar que no se ha encontrado en la literatura consultada resultados mejores, aunque estos son escasos y sólo encontramos resultados para la instancia “berlin52” con resultados peores a los obtenidos -

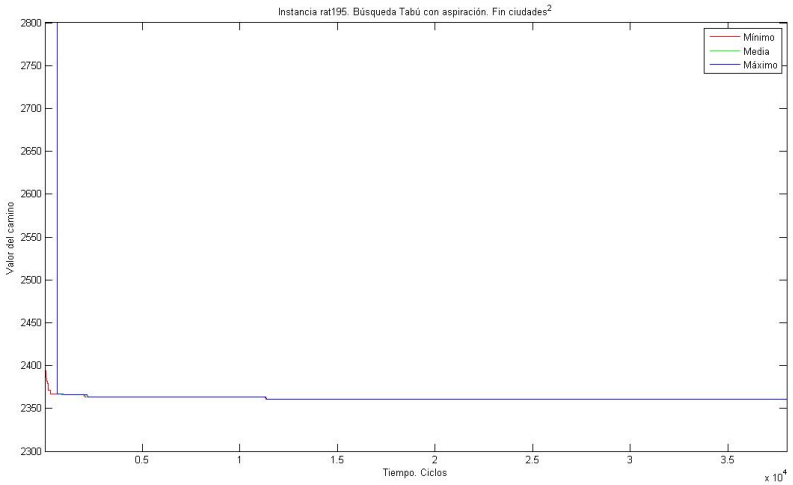


Fig. 5.23. TS rat195. Valor del tour

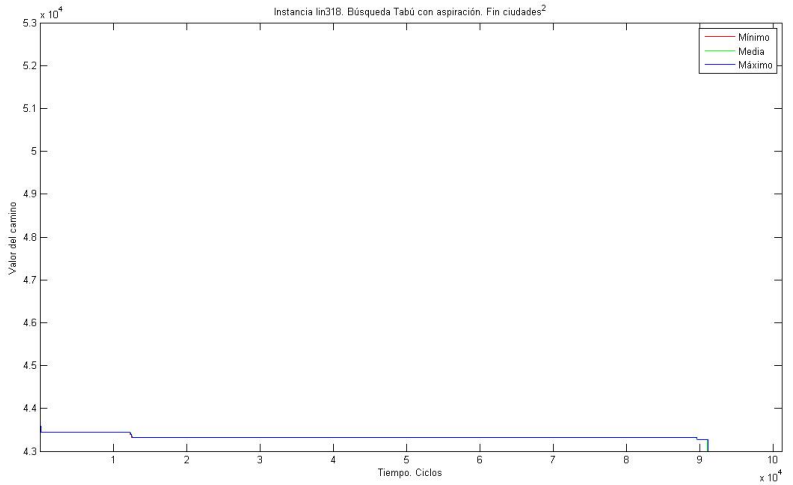


Fig. 5.24. TS lin318. Valor del tour

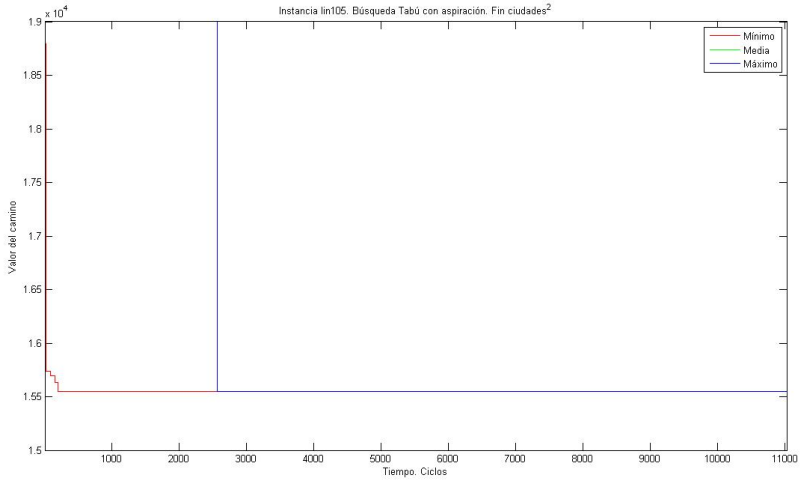


Fig. 5.25. TS lin105. Valor del tour

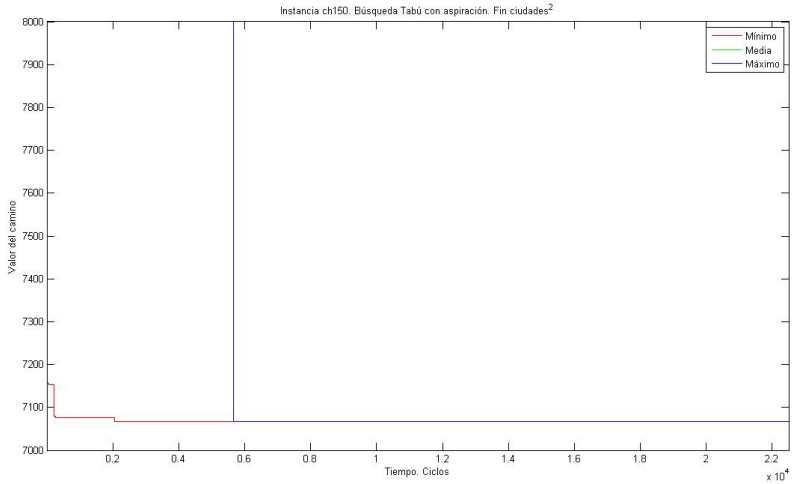


Fig. 5.26. TS lin105. Valor del tour

en nuestro experimento donde se encuentra el valor óptimo (7542). Por ejemplo, en el libro "*Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature"* de *Ke-Lin Du, M. N. S. Swamy* realizan una prueba para un versión iterativa de Búsqueda Tabú con berlin52, tras 100 iteraciones el algoritmo para con un resultado de 7782,9844.

En el artículo "*El problema del agente viajero: un algoritmo determinístico usando Búsqueda Tabú*" de *Erasmos López, Óscar Salas y Alex Murillo* consiguen un valor de 8741,00 para la instancia berlin52.

En la gráficas expuestas hemos querido visualizar dos de los mejores resultados (rat195 y lin318) y otras dos para los peores (lin105 y ch150), por si podemos inferir alguna conclusión.

La primera es que no existe una gran adaptación del algoritmo a un entorno P2P. Se parecen muchos más a las gráficas de heurísticas básicas que a las de Búsqueda Dispersa. Desde luego, la implementación aquí realizada está basada en la original, cuya única mejora añadida ha sido introducir el criterio de aspiración. Quizás incluir otras estrategias que se aplican a este método, como la diversificación, memoria a largo plazo, etc podrían redundar en un aumento de la eficiencia del mismo.

Otra conclusión que podemos observar es que o la comunicación entre nodos es muy mala o la mayoría de nodos no son capaces de mejorar sus resultados en las instancias con peores resultados; porque si vemos la línea del máximo, esta mantiene su valor máximo durante un largo periodo desde el principio, hasta que el protocolo de comunicaciones hace su trabajo y va transmitiendo a los nodos el mínimo.

Por último, el valor medio va acompañado muy de cerca durante el principio de la simulación al máximo y después igualan su valores. La primera parte no se aprecia en las gráficas porque el eje Y ha sido recortado para observar de cerca lo que ocurre en el mínimo. Esto se aprecia en las trazas que se incluye en la toda la documentación de este trabajo. Esto es una indicación de que solo unos pocos nodos son los que realmente encuentran buenos valores. Esto prácticamente no ocurre en los mejores resultados.

CAPÍTULO 6. CONCLUSIONES

6.1.- CONCLUSIONES FINALES

En este capítulo final de este trabajo fin de grado nos hemos introducido en varios aspectos muy interesantes de las Ciencias de la Computación, como el casi imposible abordaje de los problemas de optimización combinatoria por su relación con la clase de complejidad NP-ardua; el uso de metaheurísticas como herramientas para la resolución de estos problemas y aplicar las redes P2P como forma de computación distribuida. Evidentemente, todos estos paradigmas son muy amplios y complejos, objeto de innumerables artículos científicos, por lo que, lo único que se ha podido hacer desde este trabajo fin de grado, es dar acercamiento a ellos.

Este trabajo ha perseguido como objetivo principal conocer el comportamiento de Scatter Search en un entorno distribuido P2P. Sin duda, se ha realizado una implementación de este heurístico adaptado a este tipo de redes y hemos podido observar como sus resultados han sido muy buenos, tanto por la calidad como por su gran adaptabilidad a este nuevo entorno; en comparación con el resto de heurísticos que hemos probado en este trabajo.

Desde luego, estas herramientas abren un abanico muy grande de posibilidades para abordar los problemas de optimización combinatoria muy importante. Pues, permite añadir más combinaciones en los propios algoritmos como a los recursos que se comparten en la red P2P.

El simulador PeerSim despista un poco, al principio, porque no es un simulador al uso y existe un pérdida de realidad. Sin embargo, cuando se inicia en su conocimiento se comprende que su arquitectura y su uso parecido a un *framework*, resulta ser todo un acierto. El nivel de personalización y modularidad ha facilitado mucho el trabajo de implementación. Añadir nuevas funcionalidades es muy simple y presenta una magnífica predisposición para la investigación científica, puesto que, el control y la recolección de información de lo que está sucediendo durante la ejecución de la simulación es muy fácil de obtener.

6.2.- TRABAJOS FUTUROS

Como se ha comentado, los campos aquí usados son tan amplios, complejos y que aún se encuentran en investigación que este trabajo tan sólo puede introducirse en ellos. Las líneas futuras son muchas pero las más inmediatas y directamente relacionadas con este trabajo, se consideran las siguientes.

- El más evidente, es ampliar el número de metaheurísticos a probar en este entorno. Existen muchos como templado simulado, GRASP, multifragmentos, colonias de hormigas y otros algoritmos evolutivos.
- Búsqueda Tabú ha dejado un sabor amargo. Como se propone, sería interesante implementar en un futuro, las distintas variantes que se aplican al original como la diversificación, memoria a largo plazo, etc. Por otro lado, analizar por qué algunas instancias tienen un comportamiento tan malo y la mayoría de nodos no logran fácilmente un buen resultado.
- Sus autores han hecho un muy buen trabajo con PeerSim pero durante el desarrollo de este trabajo se ha echado en falta que este simulador añadiera capacidades de ejecución paralela y/o concurrente para que pudiera aprovechar los procesadores multiproceso que son hoy en día tan comunes. Verdaderamente esta función sería muy útil y mejoraría el rendimiento y los tiempos de simulación, ya que, ahora mismo todos los nodos de la red P2P tienen que ser ejecutados por el procesador en un único hilo de manera secuencial. Esta conversión de PeerSim no puede ser muy dificultosa debido a que está implementado en JAVA. Este lenguaje facilita mucho la programación multihebra. Además, la arquitectura abierta de PeerSim hace que se muy propicia esta empresa.
- Un último trabajo futuro, podría ser aprovechar las implementaciones ya realizadas para utilizarlas en implementar algunas de las aplicaciones reales del viajante de comercio como la logística, tan de moda últimamente.

CAPÍTULO 7. BIBLIOGRAFÍA

- Arnow D., Weiss G., Introducción a la Programación con Java, Addison-Wesley, 2001.
- David L. Applegate, Robert E. Bixby, Vasek Chvátal & William J. Cook, The Traveling Salesman Problem: A Computacional Study, Princenton University Press, 2006.
- Fraufe Agustín, JAVA 2, manual de usuario y tutorial, Ra-Ma Editorial, 2000.
- Guerequeta Rosa, Vallecillo Antonio, Técnicas de diseño de algoritmos, Servicio de Publicaciones de la Universidad de Málaga, 1997.
- Glover F., "A template for Scatter Search and Path Relinking in Artificial Evolution", Selected Papers from the Third European Conference on Artificial Evolution, Jin Kao-Hao et al. (eds.), Lecture Notes in Computer Science1363,Springer-Verlag, Berlin Heidelberg, 1998.
- Horowitz E., Sahni S., Fundamentals of Computer Algorithms, Computer Science Press, 1978.
- Reinelt G., The Traveling Salesman. Computational Solutions for TSP Applications, Lecture Notes in Computer Science 840, Springer Verlag, Berlin Heidelberg, 1994 .
- PeerSim P2P Simulator Homepage, http://peersim.sourceforge.net/
- TSPLIB library website, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

CAPÍTULO 8. ANEXO I.

8.1.- INTEGRACIÓN DE PeerSim EN EL IDE ECLIPSE

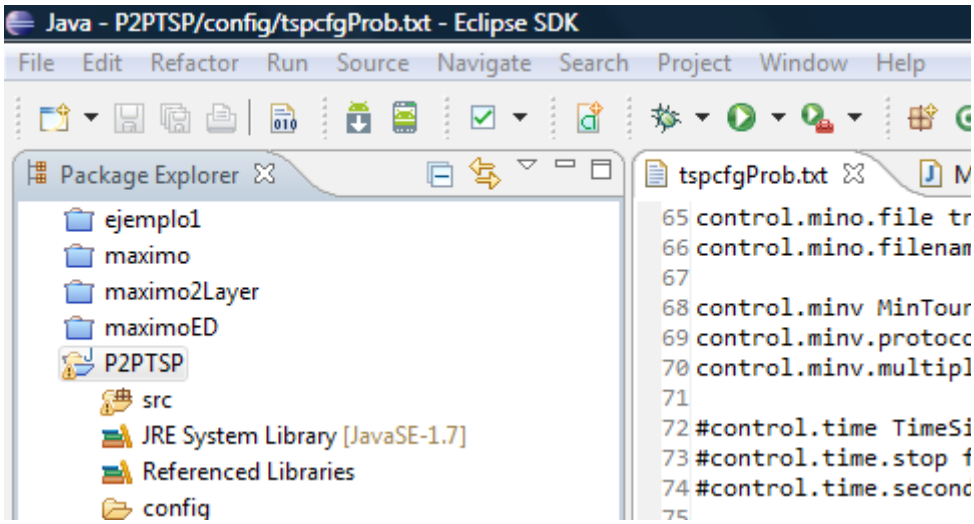
Como requisito fundamental es tener instalado en el equipo el IDE Eclipse y el JDK de JAVA.

El primer paso es descargar el software de PeerSim, se puede realizar de su página web, https://sourceforge.net/projects/peersim/ . La última versión es la 1.0.5. Probablemente no existan versiones superiores, ya que, los propios desarrolladores han anunciado el cese del proyecto (no así el de contribuciones en forma de plug-in y nuevos protocolos).

En segundo lugar, hemos de añadir estas bibliotecas a nuestro proyecto para que vayan unidas a nuestro código.

Por tanto, en Eclipse se crea un nuevo proyecto para la implementación que se vaya a realizar. *File → New → Java Project*

A la izquierda del IDE, hay un apartado que se llama el explorador de paquetes (*Package Explorer*) donde está la lista de proyectos. Ahí encontraremos el nombre del proyecto que recién hemos creado. Se hace click con el botón derecho del ratón para acceder a las propiedades del proyecto.

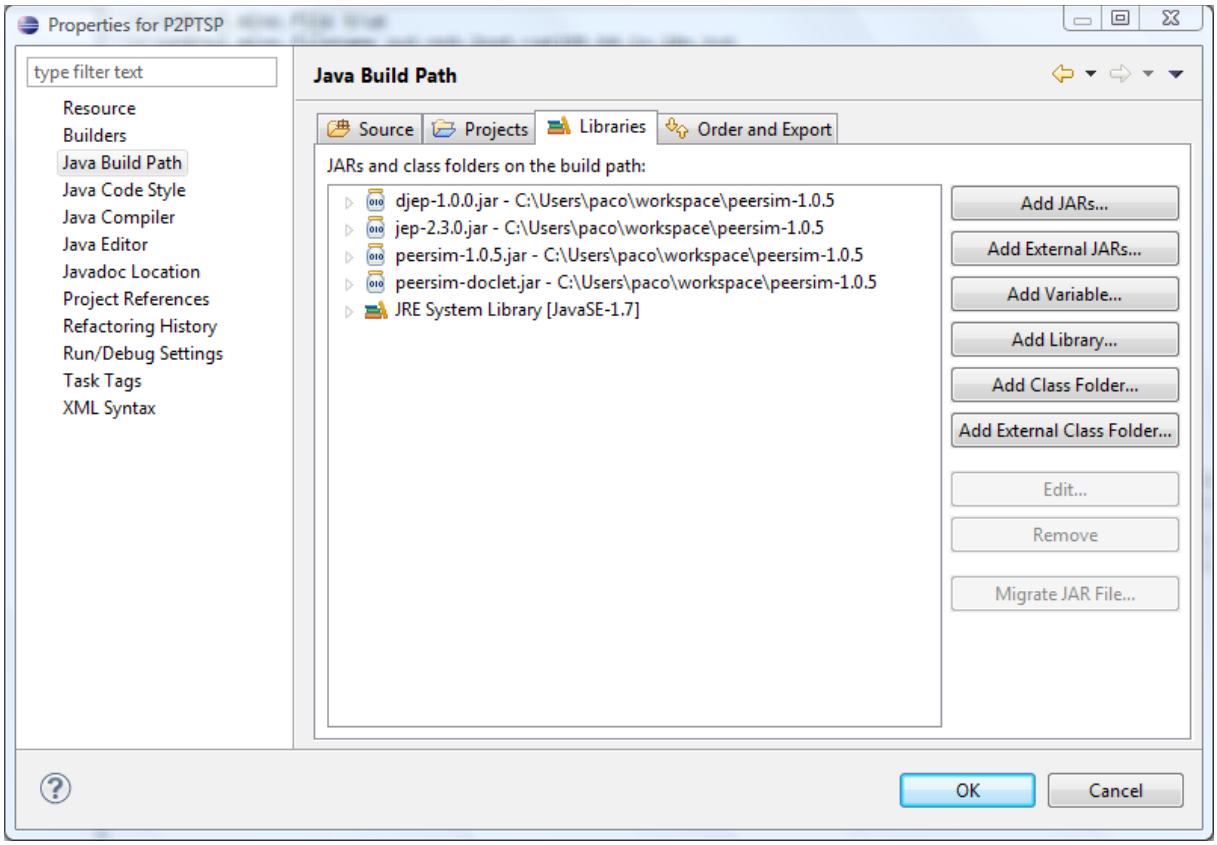


Una vez se abre la ventana de propiedades del proyecto, nos colocamos en

el apartado de "*Java Build Path*". Allí seleccionamos la pestaña "*Libraries*".

En el software de PeerSim que nos hemos descargado aparecen tres archivos de biblioteca de JAVA jar: *peersim-1.0.5.jar, jep-2.3.0.jar* y *djep-1.0.0.jar*. Estos archivos son los que tenemos que añadir al proyecto para que se produzca la llamada al motor de PeerSim y poder ejecutar la simulación desde el IDE Eclipse.

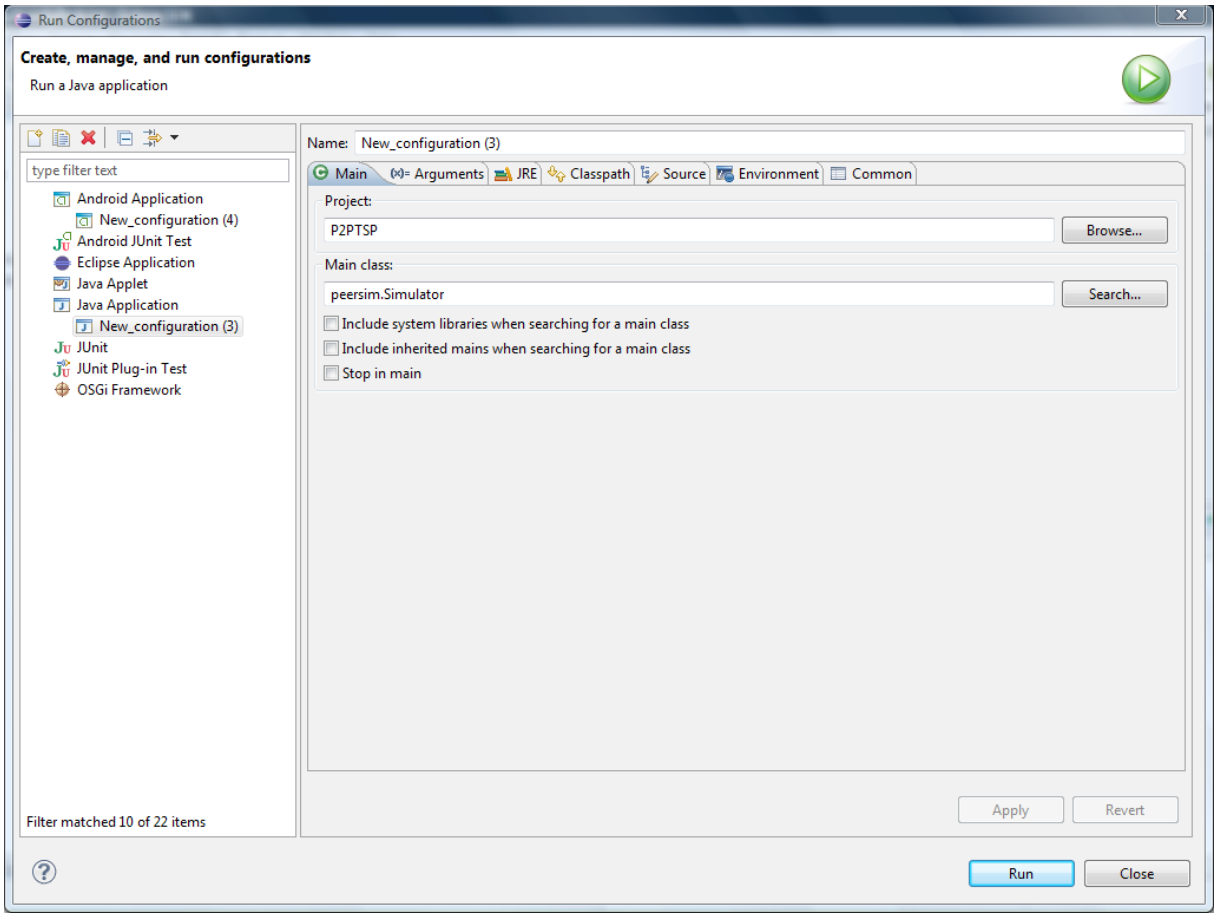
Por tanto, pulsamos el botón "*Add External JARs*" y mediante el explorador que se abre se navega por el sistema de ficheros hasta la localización de las tres bibliotecas mencionadas anteriormente.



Una vez realizado esto se pulsa el botón de OK para cerrar la ventana.

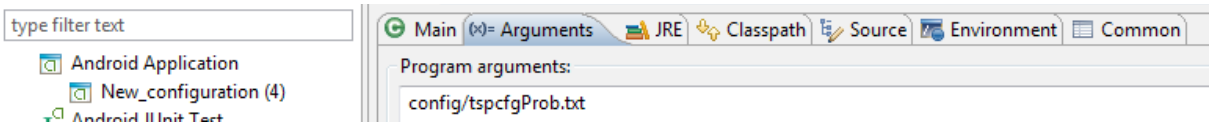
El último paso es configurar la ejecución del proyecto desde Eclipse para que se lance la simulación con los parámetros correctos, concretamente, el fichero de configuración inicial que acompaña a todas simulación de PeerSim.

En el menú *Run → Run Configuration* de Eclipse, se abre una nueva ventana. En la entrada llamada "*Java Aplication*" se crea una nueva configuración con el nombre que deseemos.



Hemos de seleccionar la clase principal del proyecto. Es decir, la que se llamará en primer lugar. Esta debe ser el motor de PeerSim. Por tanto, se pulsa el botón *Search* y se escoge *Simulator – peersim*

Por último, en la misma ventana, se elige la pestaña *Arguments.* Ahí colocaremos la ruta del fichero de configuración de la simulación para que sea pasado como parámetro en la llamada a PeerSim.



8.2.- EJECUCIÓN EN LÍNEA DE COMANDOS

En la fase de desarrollo es muy útil integrar PeerSim con el IDE para facilitarnos esta tarea.

Sin embargo, es muy interesante poder lanzar las simulaciones en línea de comandos para realizar las simulaciones fuera del ambiente del entorno de desarrollo o para lazar procesos de lotes.

La sintaxis es muy simple. Tan sólo hay que tener resuelta la variable CLASSPATH cuando instalamos JAVA y la situación de los ficheros que emplearemos en la simulación para que puedan ser visibles para la máquina virtual de JAVA.

Un ejemplo es el siguiente:

```
java -cp "jep-2.3.0.jar;djep-1.0.0.jar;p2ptsp.jar" peersim.Simulator
/home/paco/P2PTSP/config/cfg-nnh-2opt-rat195-32-1n-m2.txt
```

En primer lugar está la llamada a la máquina virtual de JAVA con el modificador *-cp* para indicarle a la máquina virtual cual son las bibliotecas que tiene que cargar. A continuación se relacionan dichas librerías que van encerradas entre comillas dobles. Como vemos están dos de las tres librerías que hemos descargado del software de PeerSim. La tercera que falta va incluida en la biblioteca que se ha generado con el software del metaheurísticos a simular *p2ptsp.jar.* Posteriormente se indica la clase principal que se llama al inicio *peersim.Simulator* y por último como parámetro de esta última, la ruta hacia el fichero de configuración.